Contenido

Práctica 4
Procesador: arquitectura, camino de datos y control

En esta sesión se trata de consolidar los conocimientos adquiridos sobre el camino de datos de un procesador y el control del mismo. Para ello utilizaremos parte del repertorio de instrucciones RISC-V[1].

El objetivo de la práctica es identificar los componentes del camino de datos que utiliza cada tipo de instrucción y el control del mismo. Así mismo, teniendo en cuenta el retardo de cada componente del camino de datos, se analiza el camino crítico para cada instrucción. El objetivo es que cualquier instrucción se pueda interpretar en un ciclo de reloj. Finalmente se determina el valor mínimo del periodo de la señal de reloj y la duración de la misma en cada uno de los dos niveles lógicos.

Descripción de la arquitectura del procesador

Todas las instrucciones del repertorio de instrucciones RISC-V que se utilizan están codificadas utilizando 4 bytes y el camino de datos del procesador es de 32 bits.

La arquitectura define 32 registros (x0, . . ., x31) de 32 bits, que son accesibles mediante campos especificados en una instrucción (banco de registros, Figura 4.1). El registro x0 es especial y siempre se lee el valor cero. Cualquier actualización del registro x0 se ignora.

El registro contador de programa (CP) no pertenece al banco de registros (Figura 4.1). Los dos bits menos significativos de este registro son siempre cero, ya que las instrucciones deben estar alineadas a cuatro bytes en el espacio lógico[2]. Este registro lo puede actualizar explícitamente una instrucción de secuenciamiento condicional o incondicional,

1. Editors: Andrew Waterman, Krste Asanovic. The RISC-V Instruction Set Manual. Volume I: User-Level ISA. Document Version 2.2. May 7, 2017.
2. El subconjunto de instrucciones implementadas.

lo puede leer una instrucción de llamada a subrutina y finalmente, lo puede leer una instrucción específica para realizar un cálculo cuyo resultado es una dirección relativa a la dirección de la instrucción.

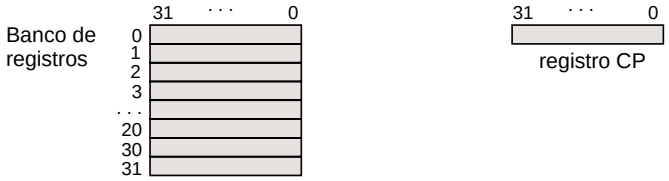


Figura 4.1 *Banco de registros y registro CP.*

El lenguaje máquina RISC-V tiene instrucciones para leer o escribir datos en memoria (MEM), instrucciones para efectuar cálculos aritmético-lógicos (CAL) e instrucciones que permiten modificar el secuenciamiento implícito (MS). En la Figura 4.2 se muestran los cuatro formatos base de las instrucciones.

	Campos de la instrucción						
	31 . . . 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0	identificador de bit
Formato	7	5	5	3	5	7	**Ejemplos de utilización**
R	funct7	rs2	rs1	funct3	rd	CoOp	CAL
I	literal12		rs1	funct3	rd	CoOp	CAL, MEM, MS
S	literal7	rs2	rs1	funct3	literal5	CoOp	MEM
U	literal20				rd	CoOp	MS

Figura 4.2 *Formato de las instrucciones de lenguaje máquina. Todas las instrucciones se codifican utilizando 4 bytes, con un campo de código de operación de 7 bits, ubicado en los bits <6:0> de la instrucción.*

Seguidamente se describen, para cada uno de los formatos base[3], ejemplos de instrucciones que los utilizan. Posteriormente se describen conjuntos de instrucciones por su funcionalidad. La descripción de todas las instrucciones consideradas en esta práctica está en el Apéndice 4.3," Especificación semántica de las instrucciones". El superíndice v en un identificador de registro indica contenido del registro.

Instrucciones con formato R. La operación de la instrucción se codifica en los campos CoOp, funct3 y funct7. El valor del campo CoOp en todas las instrucciones de tipo R es el mismo (Figura 4.3). Entonces, la operación concreta está codificada en los campos funct3 y funct7. Los operandos están codificados en los campos rs1, rs2 y rd, que se interpretan como identificadores de registro. Los campos rs1 y rs2 son identificadores de registro fuente y el campo rd es identificador de registro destino.

3. R:Register. I: Immediate. S: Source. U: Upper.

Descripción	Operandos			Especificación semántica
suma algebraica (add)	rs2	rs1	rd	$rd^v = rs1^v$ add $rs2^v$
operación or bit a bit (or)	rs2	rs1	rd	$rd^v = rs1^v$ or $rs2^v$

Figura 4.3 *Ejemplos de instrucciones con formato R. La dirección de una instrucción está almacenada en el registro CP. Entre paréntesis se muestra el nemotécnico de la instrucción.*

En el resto de formatos usualmente se utiliza un campo literal en la instrucción. El detalle de la construcción del valor inmediato que se utiliza en la operación, que especifica la instrucción, se efectúa en el Apéndice 4.1. En esta sección se utilizan los acrónimos definidos en el apéndice.

Instrucciones con formato I. La operación está codificada en el campo CoOp. Estas instrucciones utilizan como operandos fuente un registro (campo rs1) y el campo literal o inmediato especificado en la instrucción (Figura 4.4). El campo rd especifica el identificador del registro destino. El campo literal se interpreta como un número entero (complemento a dos) en todas las instrucciones. Una variante son las instrucciones de desplazamiento que utilizan el campo shamt como literal [4].

En una instrucción de llamada a procedimiento[5], la dirección de retorno, que se almacena, es la dirección de la instrucción que sigue a la instrucción de secuenciamiento en el espacio lógico. Esto es, como dirección de retorno se almacena la dirección de la instrucción de secuenciamiento más 4 bytes.

Descripción	Operandos			Especificación semántica
suma con literal (addi)	literal	rs1	rd	$rd^v = rs1^v$ add I-Imm
operación and bit a bit con literal (andi)	literal	rs1	rd	$rd^v = rs1^v$ and I-Imm
desplazamiento lógico a la izquierda (sll)	literal	rs1	rd	$rd^v = rs1^v$ << shamt
lectura de memoria (lw)	literal	rs1	rd	$rd^v = M[rs1^v + \text{I-Imm}]$
secuenciamiento incondicional indexado, con almacenamiento de la dirección de retorno (jalr)	literal	rs1	rd	$CP^v = (rs1^v + (\text{I-Imm} << 1))\|_{31:1}$ & '0' y $rd^v = CP^v + 4$

ExtSig: extensión de signo
I-Imm = ExtSig (inst(31:20))
<<: desplazamiento a la izquierda
inst: secuencia de bits que se interpreta
shamt = inst(24:20)
&: concatenación

Figura 4.4 *Ejemplos de instrucciones con formato I. Una extensión de signo hasta completar un operando de 32 bits se identifica como ExtSig. M[. . .] indica acceso a memoria. CP almacena la dirección de la instrucción.*

4. El campo shamt son los cinco bits de menor peso del campo literal12. El resto de bits del campo literal se utiliza para codificar instrucciones.
5. Secuenciamiento incondicional con almacenamiento de la dirección de retorno.

Instrucciones con formato S. Este formato se utiliza para codificar instrucciones con tres operandos fuente, siendo uno de ellos un literal y los otros dos valores almacenados en registros. Un ejemplo son las instrucciones store (Figura 4.5). Notemos que el campo literal está distribuido (Figura 4.2).

Descripción	Operandos	Especificación semántica
escritura en memoria (sw)	literal | rs2 | rs1	M [$rs1^v$ + S-Imm] = $rs2^v$
S-Imm = ExtSig (inst(31:25), inst(11:7))	inst: secuencia de bits que se interpreta	

Figura 4.5 *Ejemplo de instrucción con formato S.*

Instrucciones con formato U. Este formato se utiliza para codificar instrucciones donde un operando es un literal y el otro operando es un registro. En particular puede ser el registro CP. El resultado se almacena en un registro (Figura 4.6).

Descripción	Operandos	Especificación semántica
sumar un inmediato al registro CP (auipc)	literal | rd	rd^v = CP + (U_Imm << 12)
almacenar un literal en los 20 bits más significativos de un registro (lui)	literal | rd	rd^v = U_Imm << 12
<< 12: desplazar 12 posiciones a la izquierda	U_Imm = inst(31:12) inst: secuencia de bits que se interpreta	

Figura 4.6 *Ejemplo de instrucción con formato U.*

Adicionalmente se utilizan dos formatos más de instrucciones denominados J y B, los cuales se diferencian de los formatos básicos U y S, respectivamente, en la construcción del valor inmediato a partir del literal de la instrucción.

Instrucciones con formato B. Este formato se utiliza en las instrucciones de secuenciamiento condicional. La condición se evalúa comparando dos registros. Cuando se cumple la condición, la dirección que modifica el secuenciamiento implícito se calcula de forma relativa a la dirección de la instrucción (Figura 4.7).

Descripción	Operandos	Especificación semántica
modificar el secuenciamiento si el contenido de los registros es el mismo (beq)	literal | rs2 | rs1	if ($rs1^v$ == $rs2^v$) then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
modificar el secuenciamiento si se cumple la condición, al efectuar una comparación del contenido de los registros, interpretando el contenido como números naturales (bltu)	literal | rs2 | rs1	if ($rs1^v$ < $rs2^v$)$\|_n$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
modificar el secuenciamiento si se cumple la condición, al efectuar una comparación del contenido de los registros, interpretando el contenido como números enteros (blt)	literal | rs2 | rs1	if ($rs1^v$ < $rs2^v$)$\|_e$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
B-Imm = ExtSig (inst(31), inst(7), inst(30:25), inst(11:8)) inst: secuencia de bits que se interpreta		$\|_n$: operación con naturales $\|_e$: operación con enteros; representados en complemento a dos

Figura 4.7 *Ejemplo de instrucciones con formato B.*

Instrucciones con formato J. Este formato se utiliza en instrucciones de secuenciamiento relativo a la dirección de la instrucción. Además, la dirección de retorno se almacena en un registro especificado en la instrucción (Figura 4.8).

Descripción	Operandos	Especificación semántica
secuenciamiento incondicional relativo, con almacenamiento de la dirección de retorno (jal)	literal | rs1 | rd	$CP^V = CP^V + (J\text{-}Imm << 1)$ y $rd^V = CP^V + 4$
J-Imm = ExtSig (inst(31), inst(19:12), inst(20), inst(30:21))		inst: secuencia de bits que se interpreta

Figura 4.8 *Ejemplo de instrucción con formato J.*

Instrucciones de cálculo

Las instrucciones de cálculo especifican como operandos fuente dos registros del banco de registros o un registro del banco de registros y un literal. El resultado del cálculo se almacena en un registro del banco. Ejemplos de ellas son las instrucciones de la Figura 4.3 y las tres primeras instrucciones de la Figura 4.4.

Instrucciones de secuenciamiento

Las instrucciones de secuenciamiento condicional especifican dos registros. La condición que se evalúa es el resultado de comparar el contenido de los dos registros (rs1 y rs2). Ejemplos de instrucciones de secuenciamiento condicional e incondicional son la última instrucción de la Figura 4.4, las tres instrucciones de la Figura 4.7 y la instrucción de la Figura 4.8.

En la Figura 4.4 y la Figura 4.8 se muestran instrucciones de secuenciamiento incondicional.

También se dispone de instrucciones que, además de modificar el secuenciamiento, permiten almacenar la dirección de una instrucción, denominada dirección de retorno. El objetivo es modificar el flujo de instrucciones que se está interpretando en secuencia y volver a él, una vez se ha ejecutado un conjunto de instrucciones ubicado en otro trozo del espacio lógico (subrutinas). Ejemplos son la última instrucción de la Figura 4.4 y la instrucción de la Figura 4.8.

Instrucciones de acceso a memoria

Las instrucciones de acceso a memoria utilizan los formatos I y S. El contenido del registro fuente rs1 se suma al campo literal formateado (se interpreta en complemento a dos) para calcular una dirección efectiva que se utiliza para acceder a memoria.

En una instrucción LOAD el valor leído se almacena en el registro rd. En una instrucción STORE el contenido del registro rs2 se almacena en memoria. Ejemplos son la cuarta instrucción de la de la Figura 4.4 y la instrucción de la Figura 4.5.

Un dirección efectiva de memoria especifica la dirección de una posición de almacenamiento que contiene un byte. Las instrucciones load y store, que leen o escriben 32 bits, deben tener alineada la dirección a 4 bytes y las que leen o escriben 16 bits deben tener alineada la dirección a 2 bytes[6].

El formato en el cual se almacenan datos de tamaño mayor que un byte se denomina "little-endian" (Figura 4.9). El byte de menor ponderación de un dato se almacena en la dirección menor de las direcciones que permiten acceder a los bytes individuales del dato. El resto de bytes se almacena de forma consecutiva siguiendo la ponderación. En estas condiciones, en un acceso a memoria la dirección efectiva es la del byte de menor peso.

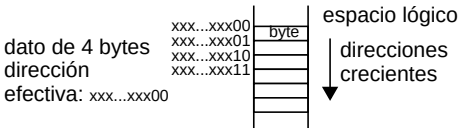


Figura 4.9 *Formato de almacenamiento "little-endian".*

Microarquitectura

En esta práctica nos centraremos en la microarquitectura de un procesador. La microarquitectura es el nivel existente entre la arquitectura o lenguaje máquina de un procesador y los elementos lógicos que se utilizan para construirla.

Para efectuar la descripción utilizaremos niveles de abstracción. Empezaremos desde el más externo para llegar al de componentes como el banco de registros y la ALU.

Primer nivel

En el nivel de abstracción más externo distinguimos la unidad procesador y los elementos de memorización de instrucciones y datos (Figura 4.10). La unidad procesador utiliza un registro denominado contador de programa (CP) para leer una instrucción almacenada en la memoria de instrucciones (MI) y la interpreta. La interpretación de la instrucción actualiza el estado del procesador[7] y en ocasiones la memoria de datos (MD). Algunas instrucciones requieren, al ser interpretadas, leer información almacenada en la memoria de datos.

6. En la especificación del lenguaje máquina RISC-V no es necesario que exista alineamiento. Ahora bien, no existe garantía de que se ejecuten atómicamente o con pérdida de rendimiento. En la implementación que se efectúa en este laboratorio debe haber alineamiento. Para que el acceso sea alineado la dirección efectiva debe ser divisible por la granularidad del acceso.

Descripción estructural de primer nivel

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity computador
        port(reloj, inicializar: in std_logic);
end;

architecture estructural of computador is
    component procesador
        port(reloj, inicializar: in std_logic;
         instr, LMD:          in std_logic_vector(31 downto 0);
        CP, dirMD, EMD:       out std_logic_vector(31 downto 0);
        MDLE:                 out std_logic_vector(.. downto 0) );
    end component;
    component memD
        port(reloj:           in std_logic;
        MDLE:                 in std_logic_vector(.. downto 0);
        dir, ED:              in std_logic_vector(31 downto 0);
        LD:                   out std_logic_vector(31 downto 0));
    end component;
    component memI
        port(dir:             in std_logic_vector(31 downto 0);
        LI:                   out std_logic_vector(31 downto 0));
    end component;

    signal CP, instr, dirMD, EMD, LMD: std_logic_vector(31 downto 0);
    signal MDLE: std_logic_vector(.. downto 0);

begin
    -- instanciación del procesador y las memorias
        PROC: procesador port map (reloj, inicializar, instr, LMD, CP, dirMD, EMD, MDLE);
        MI: memI port map (CP, instr);
        MD: memD port map (reloj, MDLE, dirMD, EMD, LMD);
end;
```

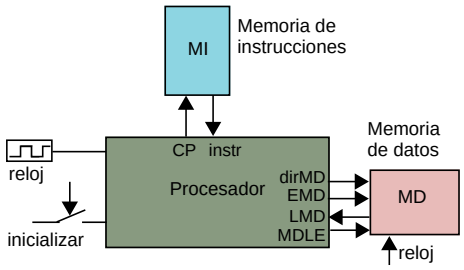


Figura 4.10 *Nivel de descripción más externo de un computador.*

Segundo nivel

En el siguiente nivel de abstracción más interno, centrándonos en la unidad procesador, se distingue el camino de datos y la unidad de control del mismo (Figura 4.11). En el camino de datos se dispone de elementos lógicos combinacionales y secuenciales para

7. El estado de un procesador está determinado por el contenido de los registros del banco de registros y el contador de programa. El estado de la arquitectura incluye la memoria de datos.

almacenar el estado del procesador y para interpretar las instrucciones. La unidad de control determina los elementos que se utilizan del camino de datos y la parte que se actualiza del estado del procesador y de la memoria de datos.

Descripción estructural de segundo nivel

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity procesador
        port(reloj, inicializar:    in std_logic;
         instr, LMD:                in std_logic_vector(31 downto 0);
        CP, dirMD, EMD:             out std_logic_vector(31 downto 0);
        MDLE:                       out std_logic_vector(.. downto 0) );
end;

architecture estructural of procesador is
    component AC
        port(reloj, inicializar:    in std_logic;
        entC:                       in std_logic_vector(... downto 0);      -- entC: señales de entrada
        condC:                      in std_logic_vector(... downto 0);      -- condC: señales de condición
        MDLE, cntC:                 out std_logic_vector(... downto 0));    -- cntC: señales de control
    end component;
    component CD
        port(reloj, inicializar:    in std_logic;
        instr:                      in std_logic_vector(31 downto 0);
        LMD, cntCD:                 in std_logic_vector(... downto 0);      -- cntCD: señales de control
        dirMD, condCD:              out std_logic_vector(... downto 0);     -- condCD: señales de condición
        CP, EMD:                    out std_logic_vector(31 downto 0);
    end component;

    signal control, condiciones: std_logic_vector(... downto 0);

begin
    control: AC port map(reloj, inicializar, instr, condiciones, MDLE, control);
    camino: CD port map(reloj, inicializar, instr, LMD, control, dirMD, condiciones, CP, EMD);
end;
```

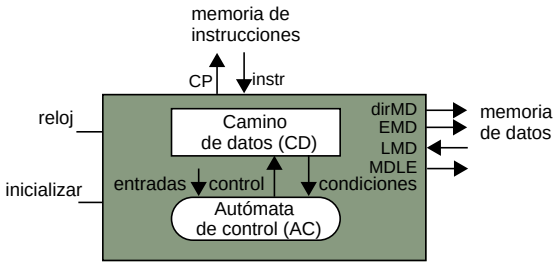


Figura 4.11 *Descripción del camino de datos (CD) y el autómata de control (AC) del procesador.*

El camino de datos contiene el registro CP y una instrucción leída de MI es una entrada en el camino de datos. Entre el camino de datos y la unidad de control existe comunicación de información en los dos sentidos. La información denominada de entrada es utilizada por la unidad de control para determinar los elementos que deben utilizarse del camino de datos. La unidad de control transmite esta información al camino de datos mediante las señales denominadas señales de control. Durante el procesado, el camino de datos también puede transmitir información a la unidad de control, para que finalice la acción de control en la interpretación de una instrucción (señales denominadas condiciones).

Observemos que el esquema descrito es un esquema genérico de un autómata que dispone de unidad de control y camino de datos. La unidad de control es un autómata de estados finitos cuyas entradas son las señales de entrada y condiciones. Las salidas del autómata son las denominadas señales de control.

Tercer nivel

En un nivel más interno de abstracción nos centramos en el camino de datos, en el cual distinguimos los siguientes elementos: a) unidad de cálculo y b) unidad de secuenciamiento (Figura 4.12). Por claridad del esquema no se muestra la distribución de las señales de reloj y de inicialización.

La unidad de secuenciamiento (US) es la encargada de determinar la dirección que se utiliza para buscar la siguiente instrucción en la memoria de instrucciones. El autómata de control se lo indica a la US mediante las señales ControlS. La US evalúa condiciones (entradasS) que se transmiten al autómata de control para determinar el secuenciamiento (condS). La salida de esta unidad actualiza el registro contador de programa (CP), el cual pertenece al estado del procesador.

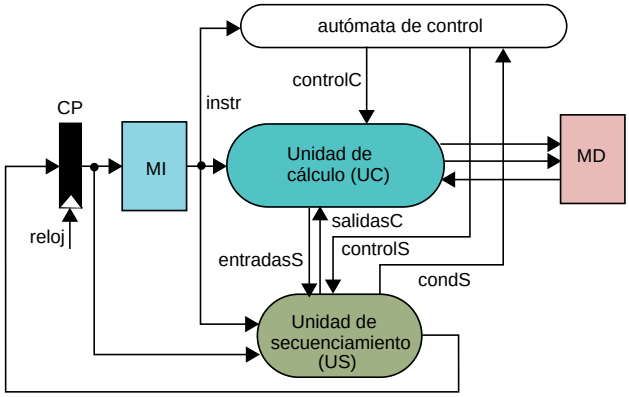


Figura 4.12 *Esquema del camino de datos donde se identifican la unidad de cálculo y la unidad de secuenciamiento. También se muestra la comunicación de información con el autómata de control.*

La unidad de cálculo se utiliza para determinar los valores con los que se actualiza el estado almacenado en el banco de registros y en la memoria de datos.

Seguidamente, en un nivel más interno de abstracción de la jerarquía de niveles, nos centramos en primer lugar en la unidad de cálculo y la parte de la unidad de control o autómata de control asociada. Posteriormente se analiza el acceso a la memoria de datos, la unidad de secuenciamiento y las organizaciones de la memoria de instrucciones y la memoria de datos.

En los esquemas de los siguientes apartados no se muestran las señales de reloj y de inicialización. Tampoco se muestra, en la mayoría de ellos, la entrada de la instrucción en el autómata de control.

Cuarto nivel: unidad de cálculo

En este apartado se describe la unidad de cálculo y la parte asociada del autómata de control (Figura 4.13). En la unidad de cálculo distinguimos los siguientes elementos: a) banco de registros (BR), b) unidad aritmético lógica (ALU), c) formateador (FMTD) y d) elementos de encaminamiento (multiplexores).

El autómata de control (AC) determina los operandos que se utilizan en el cálculo (a, b) y para ello controla los elementos de encaminamiento (mL1, mL2). Además indica a la ALU la operación que debe realizar con los operandos (opALU). Finalmente activa las señales necesarias para actualizar el banco de registros (mES, PBR).

En el autómata de control distinguiremos los siguientes componentes: a) control de la ALU (opALU) y b) control del resto de elementos.

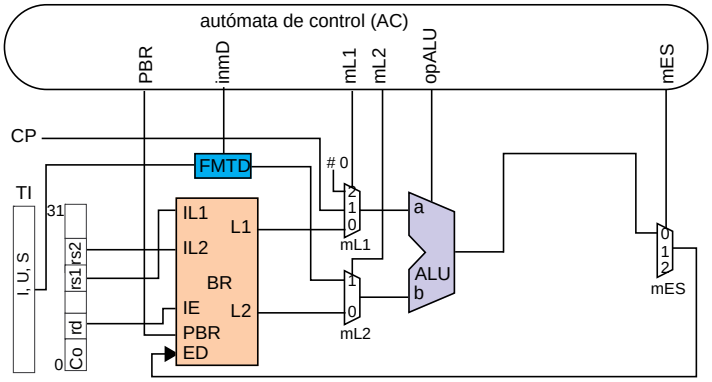


Figura 4.13 *Unidad de cálculo. Banco de registros (BR), unidad aritmético lógica (ALU) y elementos de encaminamiento. En la parte izquierda se muestran los tipos de formatos posibles en instrucciones con un campo literal. Los campos CoOp, funct3 y funct7 (o inst(31:inst(25)) de la instrucción son entradas del autómata de control (AC) y no se muestran.*

La unidad aritmético lógica, como su nombre indica, es la encargada de efectuar operaciones aritméticas y lógicas. Ejemplos de las primeras son la suma algebraica y comparaciones. Un ejemplo de la segunda es una operación lógica “and” bit a bit.

En primer lugar se describe el acceso al banco de registros y el encaminamiento de la información. Posteriormente se describen las unidades funcionales incluidas en la ALU.

Acceso al banco de registros y encaminamiento

El banco de registros, mostrado en la Figura 4.13, tiene dos caminos de lectura (L1 y L2) y un camino de escritura (ED). Las entradas de identificadores de registro asociadas son respectivamente IL1, IL2 e IE. Asociado al camino de escritura existe una señal de permiso de escritura (PBR) y la señal de reloj, que no se muestra.

Seguidamente nos centraremos en la obtención de los operandos que serán procesados en la ALU. Para ello utilizaremos un subconjunto de las instrucciones que interpreta el procesador. En la tabla de la Figura 4.14 se especifica, para el subconjunto de instrucciones, el nemotécnico utilizado en ensamblador, una descripción textual de la funcionalidad y una descripción de la operación a nivel de transferencia entre registros (RTL)[8].

FI	Nemotécnico	Descripción	Especificación semántica
R	add	suma algebraica	$rd^v = rs1^v$ add $rs2^v$
R	sll	desplazamiento lógico a la izquierda variable	$rd^v = rs1^v << rs2^v\|_{4:0}$
I	addi	suma con inmediato	$rd^v = rs1^v$ add I-Imm
I	andi	operación and bit a bit con literal	$rd^v = rs1^v$ and I-Imm
I	slli	desplazamiento lógico a la izquierda	$rd^v = rs1^v << shamt$
U	auipc	sumar un inmediato a los bits más significativos del CP	rd^v = CP + (U_Imm << 12)
U	lui	almacenar un inmediato en los bits más significativos de un registro	rd^v = U_imm << 12
		ExtSig: extensión de signo I-Imm = ExtSig (inst(31:20)). U_Imm = inst(31:12) inst: secuencia de bits que se interpreta	shamt = inst(24:20), es un natural $\|_{4:0}$: 4 bits menos significativos <<: desplazamiento lógico a la izquierda

Figura 4.14 *Subconjunto de instrucciones que se ejecutan en la unidad de cálculo. FI: formato de la instrucción.*

Los identificadores de registros siempre ocupan la misma posición en el formato de la instrucción y se interpretan de la misma forma, si es el caso.

En el acceso al banco de registros un objetivo, si no tenemos en cuenta el consumo energético, es utilizar directamente los campos de la instrucción para efectuar la fase de lectura del banco de registros. Esto es, que no sea necesario esperar alguna señal del decodificador (autómata de control). En este sentido distinguimos dos grupos de instrucciones que realizan operaciones aritméticas o lógicas. Las instrucciones cuyos dos operandos se encuentran almacenados en registros del banco de registros (tipo R) y las instrucciones donde uno de los operandos es un literal especificado en la instrucción (tipo I).

8. RTL: Register Transfer Level.

En las figuras que se muestran seguidamente se detalla el encaminamiento de las señales. Inicialmente no se muestran todas las señales de entrada de un elemento. Estas señales se añadirán a medida que se describe el subconjunto de instrucciones. Además, en las figuras se utilizan dos tonos de trazado (difuso y nítido) para distinguir el encaminamiento añadido (trazo nítido) del encaminamiento descrito previamente (trazo difuso).

La instrucción “add” utiliza como identificadores de registros fuente rs1 y rs2. La instrucción “addi” utiliza como identificador de registro fuente rs1 y el otro operando es el campo literal especificado en la instrucción, con el cual se construye el tipo de inmediato I.

El campo rs1 es entrada en el puerto IL1 y el campo rs2 es entrada en el puerto IL2 (parte izquierda de la Figura 4.15). El campo literal debe interpretarse en complemento a dos. Por ello, antes de utilizarlo como operando hay que formatearlo (FMTD). El valor debe convertirse a una representación en 32 bits, que es el número de bits de los operandos.

La salida en la fase de lectura del banco de registros es L1 y L2 o el literal formateado. Entonces, después del puerto L2 es necesario un multiplexor (mL2) para poder efectuar una selección en función de la instrucción (parte derecha de la Figura 4.15).

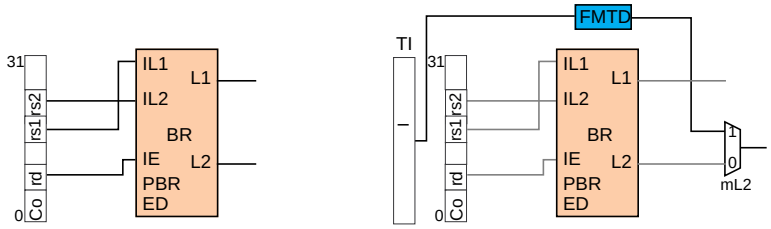


Figura 4.15 *Encaminamiento de los identificadores de registro y salida de la fase de lectura del banco de registros. En la izquierda de cada esquema se muestra el formato de las instrucciones analizadas. En la parte izquierda de la figura se muestra la instrucción “add” y en la parte derecha de la figura se muestran las instrucciones “add” y “addi”.*

La instrucción “sll” utiliza los campos rs1 y rs2 de la instrucción como identificadores de registro fuente. El campo rs1 es entrada en el puerto IL1 y el campo rs2 es entrada en el puerto IL2. Del valor leído en este último puerto, en la ALU, sólo se utilizan los cinco bits menos significativos. Este filtrado se efectúa en la ALU. El camino de datos utilizado se muestra en la parte izquierda de la Figura 4.15.

La instrucción “slli” utiliza como identificador de un registro fuente el campo rs1 y el otro operando es el campo shamt especificado en la instrucción (Figura 4.14). Para leer el operando del banco de registros utilizaremos el puerto IL1. Por otro lado, el campo shamt es un subconjunto de los bits utilizados para especificar el campo literal en la instrucción “addi” (Figura 4.15 derecha)[9]. Para suministrar este operando a la ALU utilizaremos el multiplexor (mL2) cuyas entradas son el puerto L2 y la salida de FMTD (parte derecha de la Figura 4.15).

La instrucción “auipc” utiliza como un operando fuente el contenido del registro CP (contador de programa), el cual almacena la dirección de la instrucción. Respecto al otro operando hay que formatear el campo literal para construir el inmediato de tipo U (Figura 4.14). Este formateo es distinto del utilizado en las instrucciones previas (addi y slli). Por tanto, son necesarias señales de control en la lógica FMTD.

En la fase de lectura del banco de registros, el inmediato se encamina de la misma forma que en la instrucciones previas. En consecuencia, para encaminar el contenido del registro CP es necesario un multiplexor (mL1), después del puerto L1, para poder efectuar la selección en función de la instrucción (parte izquierda de la Figura 4.16).

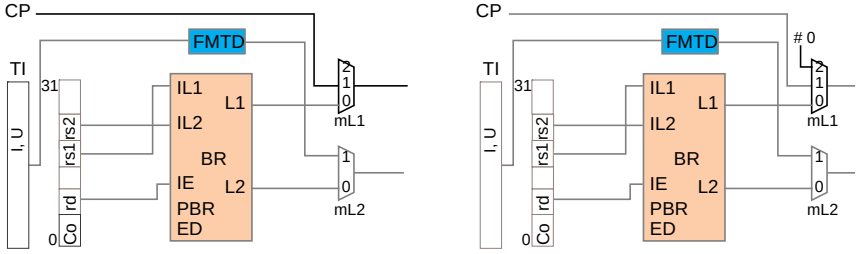


Figura 4.16 *Encaminamiento añadido para las instrucciones “auipc” y “lui”.*

En la instrucción “lui”, el campo literal se formatea de la misma forma que en la instrucción “auipc” (Figura 4.14). En la ALU se efectuará la operación suma. En consecuencia, el otro operando debe tener el valor cero. Para suministrar este valor a la ALU se añade otra entrada al multiplexor mL1 (parte derecha de la Figura 4.16).

Actualización del banco de registros. El resultado de las instrucciones de la Figura 4.14 se utiliza para actualizar el banco de registros (Figura 4.17).

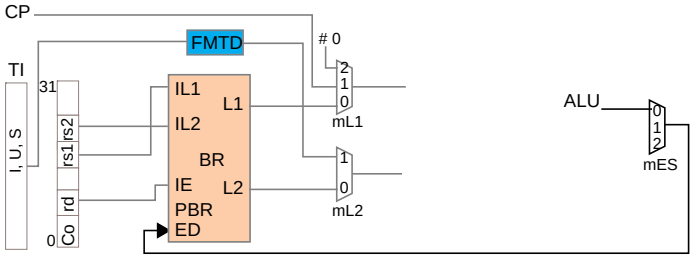
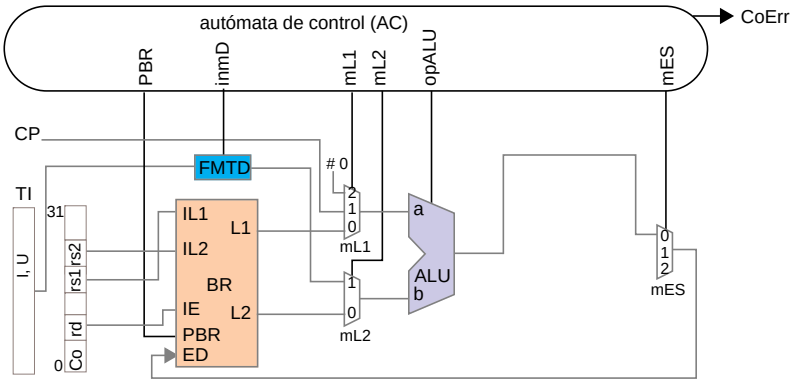


Figura 4.17 *Encaminamiento del resultado producido por la ALU para actualizar el banco de registros.*

9. El resto de bits se utiliza para decodificar las instrucciones. En el Apéndice 4.1 y en el Apéndice 4.3 se describe el detalle de la codificación. En la ALU solo se utilizan los 5 bits menos significativos. El resto de bits puede tener cualquier valor. Por tanto, no es necesario distinguir entre el formateo del literal en la instrucción “addi” y el formateo del literal en la instrucción “slli”.

Unidad de formateo (FMTD). La unidad FMTD prepara el literal especificado en una instrucción para ser utilizado como un operando en la ALU (tipo de inmediato). El rango de los valores de entrada en la ALU es mayor que el disponible en el campo literal. En todos los casos el literal debe interpretarse en complemento a dos. Por tanto, se replica el signo del campo literal para disponer de un operando de 32 bits (“addi” y “slli”, tipo I de inmediato, Figura 4.14). En una instrucción “auipc” o “lui”, que es de tipo U, y se describe en la Figura 4.14, se añaden ceros por la derecha hasta completar 32 bits. Esto es, el literal se desplaza 12 posiciones a la izquierda. Por tanto, para distinguir entre tipo de inmediatos son necesarias señales de control en la lógica FMTD.

Señales de encaminamiento y control. En la tabla de la Figura 4.18 se muestran las señales de control de los multiplexores y las señales de control de la lógica FMTD[10], que genera el autómata de control al decodificar las instrucciones. En la tabla se utilizan acrónimos para indicar la activación o no de una señal. De la misma forma, se utilizan acrónimos para indicar la entrada seleccionada en un multiplexor[11]. Respecto a la ALU solo se indica si es necesario utilizarla para ejecutar la instrucción. Una entrada vacía en las tablas indica que el valor de la señal no es relevante.



Nem.	PBR	FMTD	mL1	mL2	mES	opALU
add	si		reg	reg	ALU	si
sll	si		reg	reg	ALU	si
addi	si	I	reg	inm	ALU	si
andi	si	I	reg	inm	ALU	si
slli	si	I	reg	inm	ALU	si
auipc	si	U	CP	inm	ALU	si
lui	si	U	cero	inm	ALU	si

Figura 4.18 *Señales de encaminamiento y señales de control y error. La señal CoErr se activa si alguno de los campo que se decodifica es erróneo.*

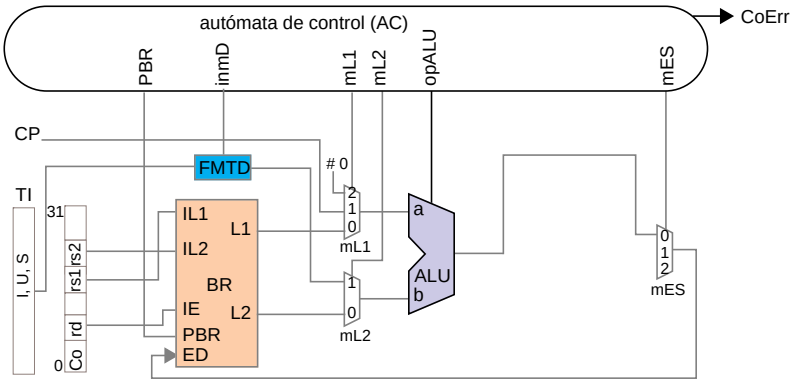
Unidad aritmético-lógica (ALU)

La ALU implementa, mediante varias unidades lógicas, un conjunto de operaciones aritméticas y lógicas (Figura 4.19). La operación aritmética básica que realiza la ALU es la suma algebraica[12].

10. El autómata de control también genera una señal de error (CoErr) cuando la secuencia de bits que se interpreta no se corresponde con una de las instrucciones implementadas.
11. La codificación binaria de cada acrónimo se muestra en el Apéndice 4.4.
12. No se detecta desbordamiento. Si es necesaria su detección hay que añadir instrucciones antes o después de la instrucción para efectuar la detección.

Otras operaciones que realiza la ALU son: a) comparación de valores, b) operación lógica y c) desplazamiento lógico o aritmético a la derecha o a la izquierda (como máximo 31 posiciones).

Instrucciones de comparación. El lenguaje máquina dispone de instrucciones (slt, sltu, slti, sltiu) para comparar el contenido de dos registros, interpretando el contenido en complementos a dos (enteros) o como números naturales (Apéndice 4.3). El resultado de la evaluación se almacena en un registro.



	opALU				
Nem.	4	3	2	1	0
add	si	suma			
sll	si	despl_izquier			
slt	si	cmp_menor_ent			
sltu	si	cmp_menor_nat			
slli	si	despl_izquier			
addi	si	suma			
andi	si	and_bit_a_bit			
auipc	si	suma			
lui	si	suma			

Figura 4.19 *Unidad funcional ALU y señales de control.*

Señales de control de la ALU. La ALU requiere de señales de control para determinar en cada unidad funcional la operación que debe efectuarse y señales para encaminar las salidas de las unidades funcionales a la salida de la ALU. En la tabla de la Figura 4.19 se muestran las señales de control utilizadas[13]. El bit más significativo indica que se utiliza la ALU para ejecutar la instrucción. Este bit se corresponde con el indicado en la Figura 4.18. Para indicar la operación concreta se utilizan los mismos bits que están especificados en la instrucción (Apéndice 4.6). En el caso particular de las instrucciones “auipc” y “lui” se indica que debe realizarse una suma.

Cuarto nivel: acceso a la memoria de datos

La memoria de datos es un elemento donde se almacenan datos que se pueden leer y actualizar. En este apartado se describe el encaminamiento de un subconjunto de las instrucciones que acceden a la memoria de datos (Figura 4.20).

13. Para indicar la operación efectuada se utilizan acrónimos. La codificación binaria de los mismos se muestra en el Apéndice 4.6.

FI	Nemotécnico	Descripción	Especificación semántica
I	lw	lectura de palabra	$rd^v = M\ [rs1^v + I\text{-}Imm]$
S	sw	almacenamiento de palabra	$M\ [rs1^v + S\text{-}Imm] = rs2^v$
		ExtSig: extensión de signo I-Imm = ExtSig (inst(31:20)).	S-Imm = ExtSig (inst(31:25), inst(11:7)) inst: secuencia de bits que se interpreta

Figura 4.20 *Subconjunto de instrucciones de acceso a la memoria de datos. M[. . .] indica acceso a memoria. La expresión que sustituye a los puntos calcula la dirección efectiva.*

Las instrucciones de acceso a memoria utilizan el banco de registros para obtener uno de los operandos, con el cual se calcula la dirección efectiva. El otro operando se obtiene a partir del literal especificado en la instrucción. Las instrucciones STORE obtienen el valor que se almacena en memoria del banco de registros. Para calcular la dirección efectiva se utiliza el sumador de la ALU. El valor calculado se utiliza posteriormente para acceder a la memoria de datos (Figura 4.21)[14].

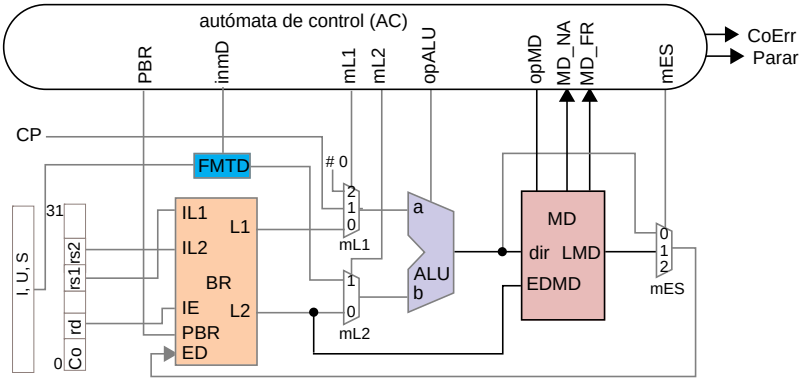


Figura 4.21 *Acceso a la memoria de datos. En la memoria no se muestra la entrada de la señal de reloj.*

El autómata de control (AC) genera las señales opMD y procesa las señales MD_FR (dirección fuera de rango) y MD_NA (dirección no alineada). La activación de estas últimas señales determina que el AC active la señal parar.

14. En el apartado “Segundo nivel: organización de la memoria de datos, en la página 254” se detalla la organización de los elementos incluidos en la MD para servir accesos de distinta granularidad (Tipos de instrucciones de acceso a la memoria de datos, en la página 241).

Acceso al banco de registros y encaminamiento

Seguidamente se describe la fase de lectura del banco de registros para un subconjunto de las instrucciones de acceso a memoria (Figura 4.20).

En una instrucción “sw” son necesarios tres operandos. Dos operandos para calcular la dirección efectiva y un tercer operando, cuyo valor se almacena en la memoria de datos. Los operandos para calcular la dirección efectiva son el contenido del registro identificado por el campo rs1 y el campo literal (offset) de la instrucción. El campo rs1 es entrada en el puerto IL1. El operando literal se suministra a la ALU utilizando el multiplexor mL2 como en una instrucción “addi” (Figura 4.15). El operando que se almacena en memoria es el contenido del registro cuyo identificador se almacena en el campo rs2 de la instrucción. El campo rs2 es entrada en el puerto IL2 (Figura 4.22).

En una instrucción “lw” son necesarios dos operandos para calcular la dirección efectiva. Estos operandos son el contenido del registro identificado por el campo rs1 y el campo literal de la instrucción. El campo rs1 es entrada en el puerto IL1. El campo rd de la instrucción es el identificador del registro destino y es entrada del puerto IE.

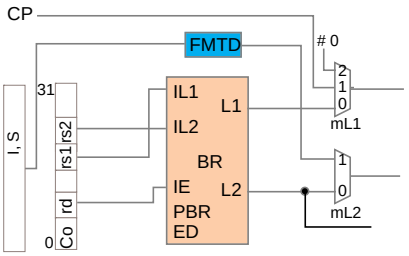


Figura 4.22 *Encaminamiento de los identificadores de registro y salida de la fase de lectura del banco de registros de las instrucciones previas y las instrucciones “sw” y “lw”. En la izquierda del esquema se muestra el formato de las instrucciones analizadas.*

Actualización del banco de registros. El resultado de las instrucciones mostradas en la Figura 4.14 y en la Figura 4.20 puede producirse en la ALU o al leer de la memoria de datos. En consecuencia, es necesario un multiplexor que permita seleccionar entre estas dos fuentes de datos para actualizar el banco de registros (multiplexor mES en la Figura 4.23).

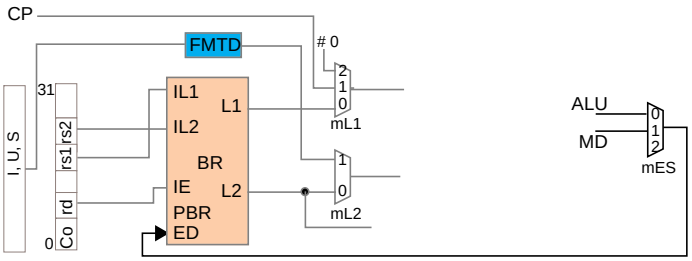
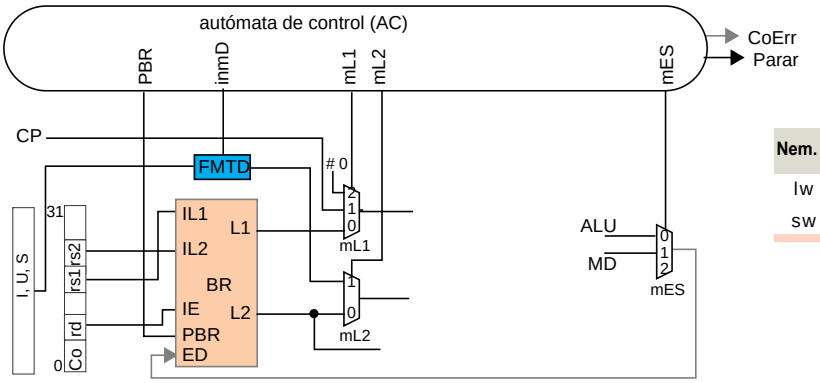


Figura 4.23 *Encaminamiento del resultado producido por la ALU y MD para actualizar el banco de registros.*

Señales de encaminamiento y control. En la tabla de la Figura 4.24 se muestran las señales de control de los multiplexores y las señales de control de la lógica FMTD, que genera el autómata de control al decodificar las instrucciones de acceso a memoria[15].



Nem.	PBR	FMTD	mL1	mL2	mES
lw	si	I	reg	inm	MD
sw	no	S	reg	inm	

Figura 4.24 *Señales de encaminamiento y señales de control para las instrucciones de acceso a memoria. La señal PBR es el permiso de escritura en el banco de registros.*

Señales de control de la ALU. Una instrucción de acceso a memoria requiere que la ALU efectúe la suma de las entradas. Por ello, las señales de control de la ALU son las mismas que en una instrucción add[16]. En la Figura 4.25 se muestran las señales de control.

15. La codificación binaria de los acrónimos se especifica en el Apéndice 4.4.
16. La codificación binaria de los acrónimos se especifica en el Apéndice 4.6.

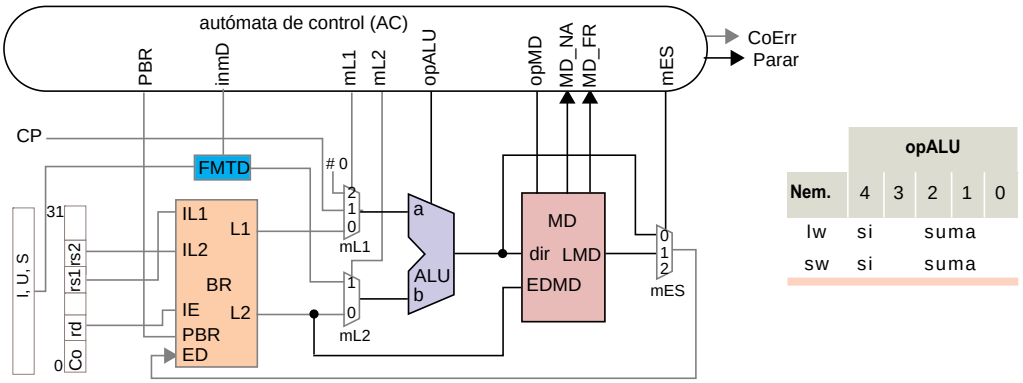


Figura 4.25 *Control de la ALU en instrucciones de acceso a memoria.*

Tipos de instrucciones de acceso a la memoria de datos

En la tabla de la Figura 4.26 se muestran las instrucciones de acceso a memoria. Se distinguen varios tipos de instrucciones en función del tamaño del dato[17] que se lee o escribe. También se distinguen varios tipos, en función de cómo se interpreta el valor leído de la memoria de datos (entero o natural).

La dirección efectiva debe estar alineada a 4 bytes cuando se leen 32 bits y a 2 bytes cuando se leen 16 bits. Si este no es el caso hay que generar una condición de error (MD_NA)[18]. También se comprueba si la dirección está dentro del rango de direcciones válidas. Si no es el caso hay que generar una condición de error (MD_FR).

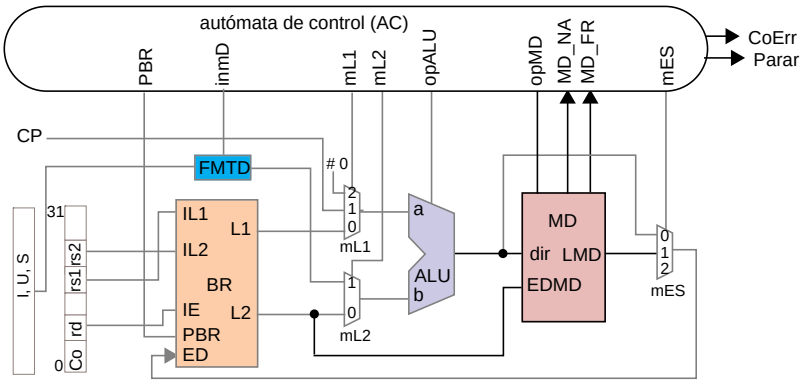
17. Granularidad del acceso.
18. En el lenguaje máquina RISC-V se indica que los accesos pueden ser no alineados. En la implementación efectuada los accesos deben estar alineados.

FI	Nemotécnico	Descripción	Especificación semántica
I	lb	lectura de un byte (como entero)	rd^v = ExtSig ((M [$rs1^v$ + I-Imm])$\|_{7:0}$)
I	lh	lectura de media palabra (dos bytes, como entero)	rd^v = ExtSig ((M [$rs1^v$ + I-Imm])$\|_{15:0}$)
I	lw	lectura de una palabra (como entero)	rd^v = M [$rs1^v$ + I-Imm]
I	lbu	lectura de un byte (como natural)	rd^v = ExtCero ((M [$rs1^v$ + I-Imm])$\|_{7:0}$)
I	lhu	lectura de media palabra (como natural)	rd^v = ExtCero ((M [$rs1^v$ + I-Imm])$\|_{15:0}$)
S	sb	almacenamiento de un byte	M [$rs1^v$ + S-Imm] = $rs2^v\|_{7:0}$
S	sh	almacenamiento de media palabra	M [$rs1^v$ + S-Imm] = $rs2^v\|_{15:0}$
S	sw	almacenamiento de una palabra	M [$rs1^v$ + S-Imm] = $rs2^v$
		ExtSig: extensión de signo ExtCero: extensión con ceros inst: secuencia de bits que se interpreta	$\|_{b..a}$: secuencia consecutiva de bits I-Imm = ExtSig (inst(31:20)) S-Imm = ExtSig (inst(31:25), inst(11:7))

Figura 4.26 *Instrucciones de acceso a memoria. Los subíndices indican los bits de un vector de bits de 32 bits que se leen o escriben. Una extensión de signo hasta completar un operando de 32 bits se identifica como ExtSig. Añadir cero en los bits más significativos hasta completar un operando de 32 bits se identifica como ExtCero. La especificación* $|_{b...a}$ *indica una secuencia contigua de bits identificada por los ordinales b y a.*

Señales de control de la memoria de datos. En la Figura 4.27 se muestran las señales necesarias para controlar el acceso a la memoria de datos. Las señales deben especificar el tipo de acceso (lectura o escritura, PMD), la granularidad o tamaño del acceso (tam) y la forma de extender, a 32 bits, el dato leído cuando sea necesario (EnNt)[19]. El bit más significativo de la señal opMD indica que es una instrucción de acceso a memoria.

19. La codificación de los acrónimos se especifica en el Apéndice 4.7.



	opMD (3... 0)					
	Ac	PMD	EnNt	tam		salida
Nem.	4	3	2	1	0	MD_NA
lb	si	L	ent	byte		
lh	si	L	ent	media_p		(1)
lw	si	L	ent	palabra		(1)
lbu	si	L	nat	byte		(1)
lhu	si	L	nat	media_p		(1)
sb	si	S		byte		
sh	si	S		media_p		(1)
sw	si	S		palabra		(1)

Figura 4.27 *Señales de control de la memoria de datos. Señal de error: (1) la MD activa la señal MD_NA cuando, en un acceso a memoria, se detecta que el acceso no está alineado. La activación de esta señal determina que el autómata de control también active la señal parar.*

Cuarto nivel: unidad de secuenciamiento

La unidad de secuenciamiento es la encargada de determinar la dirección de la siguiente instrucción que debe interpretarse.

Descripción funcional

El procesador dispone de un secuenciamiento por defecto o implícito y de un secuenciamiento explícito que se efectúa al interpretar una instrucción de secuenciamiento.

El secuenciamiento implícito calcula la dirección de instrucciones almacenadas en posiciones consecutivas de memoria. Como el tamaño de una instrucción es fijo e igual a cuatro bytes[20], es suficiente sumar cuatro a la dirección de una instrucción para calcular la dirección de la siguiente instrucción.

El valor de la dirección de una instrucción se almacena en un registro denominado CP. En la Figura 4.28 se muestra el registro CP, la lógica para la actualización explícita del mismo (USE) y la circuitería para el secuenciamiento implícito (sumador). Además se muestra la relación del retardo de la lógica con la señal de reloj. En la parte superior de la figura se muestra la señal de reloj.

20. Subconjunto de instrucciones RISC-V implementadas.

Debajo de la señal de reloj se observan registros tanto en el primer flanco ascendente como en el segundo flanco ascendente. Notemos también que el nombre de los registros es el mismo. Un registro que se muestra en el primer flanco se analiza desde el punto de vista del valor de la señal que almacena y que se observa en su salida. Un registro que se muestra en el segundo flanco se analiza desde el punto de vista del valor de la señal en la entrada del registro. Esta entrada se almacenará en el registro en el flanco ascendente.

Debajo de la señal de reloj y en la parte izquierda de la Figura 4.28 se muestra el registro CP. La salida de este registro se utiliza para acceder a la memoria de instrucciones (MI) y posteriormente interpretar la instrucción en la USE. En este apartado nos centramos en la USE que gestiona el secuenciamiento explícito. Debajo de la lógica etiquetada como MI/AC/BR/USE se muestra la lógica utilizada para efectuar el secuenciamiento implícito. El resultado de sumar cuatro a la dirección de la instrucción es la dirección de la siguiente instrucción en secuencia. Para seleccionar entre seguir en secuencia o modificar el secuenciamiento, el cual ha sido determinado por la USE, se utiliza un multiplexor. El valor en la salida del multiplexor se transfiere a la salida del registro CP en el siguiente flanco ascendente.

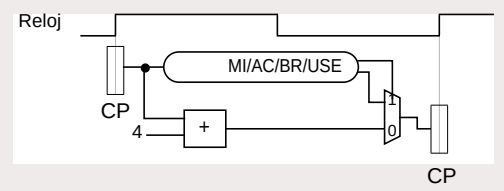


Figura 4.28 *Secuenciamiento implícito. AC: autómata de control. BR: banco de registros. MI: memoria de instrucciones. USE: unidad de secuenciamiento explícito. La unidad de secuenciamiento (US) incluye la unidad USE y el sumador (secuenciamiento implícito).*

La semántica de todas las instrucciones de secuenciamiento indica que la instrucción que debe interpretarse después de una instrucción de secuenciamiento es la que determina la instrucción de secuenciamiento.

En la Figura 4.29 se muestra de forma más detallada la unidad de secuenciamiento. En la unidad de secuenciamiento podemos distinguir: a) el registro CP, la lógica para su actualización y c) el autómata de control.

En la lógica de actualización se distinguen: a) sumadores, b) un circuito para formatear el campo literal (offset, FMTS) y c) un circuito para evaluar condiciones (EVAL). El circuito DECS es parte del autómata de control (AC) y se utiliza para determinar el sentido del secuenciamiento en instrucciones de secuenciamiento condicional. Las señales ig, me y meu indican el resultado de la condición evaluada en el módulo EVAL de la unidad de secuenciamiento. Estas señales (de condición) se utilizan en el autómata de control (DECS) para determinar el secuenciamiento.

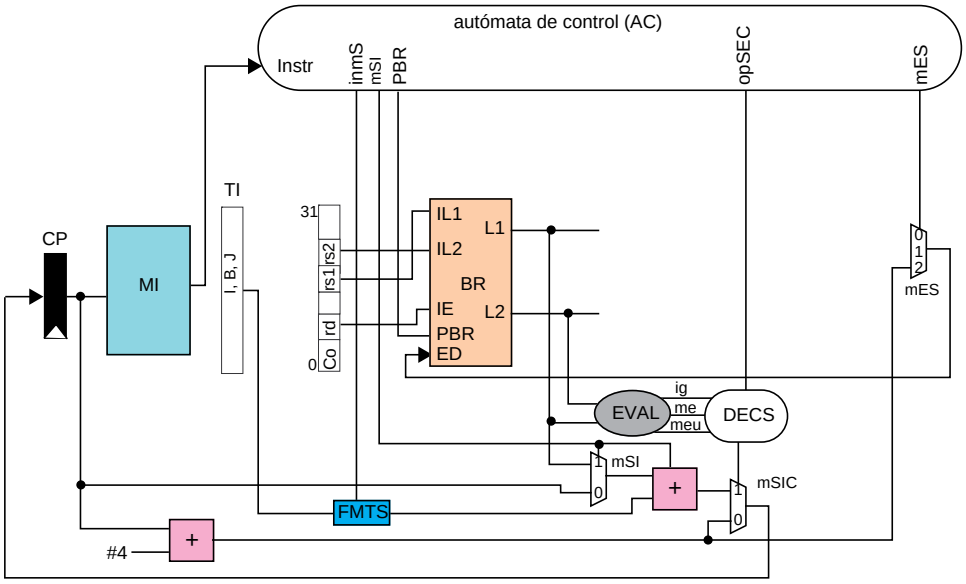


Figura 4.29 *Unidad de secuenciamiento.*

Secuenciamiento implícito

La circuitería básica de secuenciamiento incluye el registro CP y un sumador.

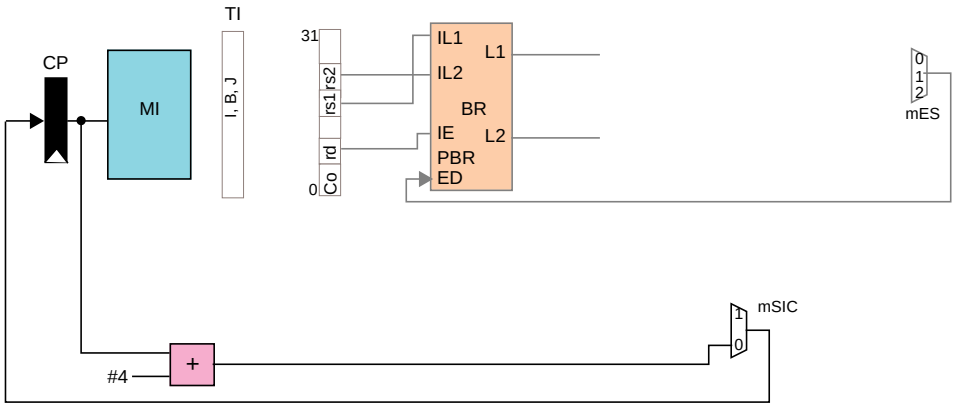


Figura 4.30 *Encaminamiento para el secuenciamiento implícito.*

Secuenciamiento explícito

Seguidamente analizaremos un subconjunto de instrucciones de secuenciamiento, con el objetivo de diseñar la lógica que se utiliza para actualizar el registro CP (tabla de la Figura 4.31).

FI	Nemotécnico	Descripción	Especificación semántica
B	beq	secuenciamiento condicional relativo	if ($rs1^v$ == $rs2^v$) then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
B	blt	secuenciamiento condicional relativo. Comparación menor que, interpretando los valores como enteros (complemento a dos)	if ($rs1^v$ < $rs2^v$)$\|_e$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
B	bltu	secuenciamiento condicional relativo. Comparación menor que, interpretando los valores como naturales (binario)	if ($rs1^v$ < $rs2^v$)$\|_n$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
I	jalr	secuenciamiento incondicional indexado. Almacenamiento de la dirección de retorno	CP^v = ($rs1^v$ + (I-Imm))$\|_{31:1}$ & '0' y rd^v = CP^v + 4
J	jal	secuenciamiento incondicional relativo. Almacenamiento de la dirección de retorno	CP^v = CP^v + (J-Imm << 1) y rd^v = CP^v + 4
	inst: secuencia de bits que se interpreta ExtSig: extensión de signo B-Imm = ExtSig (inst(31), inst(7), inst(30:25), inst(11:8))		J-Imm = ExtSig (inst(31), inst(19:12), inst(20), inst(30:21)) I-Imm = ExtSig (inst(31:20) << 1: el imm es múltiplo de 2 $\|_e$: comparación de enteros. $\|_n$: comparación de naturales. &: concatenación.

Figura 4.31 *Subconjunto de instrucciones de secuenciamiento. El símbolo & indica concatenación. El símbolo "<< 1" indica desplazar una posición a la izquierda y el símbolo '0' indica un cero en binario.*

Secuenciamiento condicional relativo. La instrucción "beq" es una instrucción de secuenciamiento condicional. La condición que se utiliza, para determinar el secuenciamiento, es el resultado de comparar si el contenido de dos registros es igual (EVAL). Los identificadores de los registros se obtienen de los campos r1s y rs2 de la instrucción. El campo rs1 es una entrada del puerto IL1 y el campo rs2 es entrada del puerto IL2 (Figura 4.32). Si no se cumple la condición se sigue en secuencia. Si se cumple la condición, la dirección destino es la suma de la dirección de la instrucción de secuenciamiento más el desplazamiento especificado en la instrucción multiplicado por dos. Previamente, el valor numérico que representa el campo inmediato, interpretándolo en complemento a dos, se representa con 32 bits (FMTS).

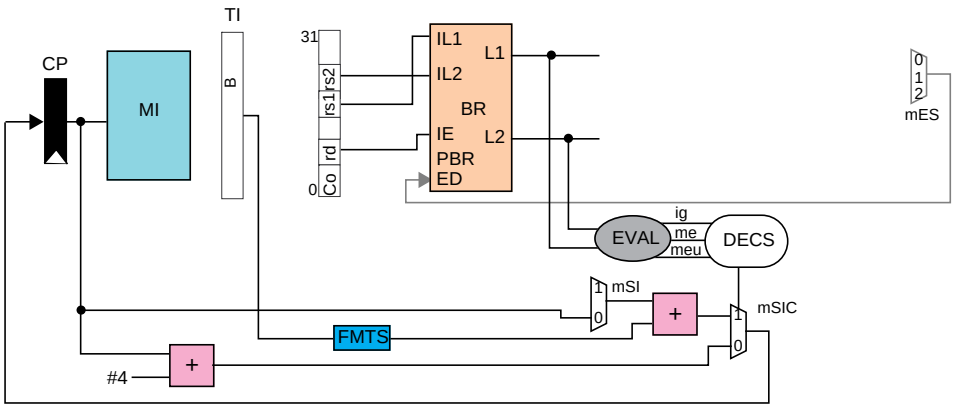


Figura 4.32 *Instrucción de secuenciamiento condicional beq.*

Circuito de evaluación de la condición (EVAL). Este circuito realiza la comparación del contenido de dos registros y genera las señales ig (igualdad), me (menor que, interpretando los valores como enteros) y meu (menor que, interpretando los valores como naturales). En la Figura 4.33 se muestra, para un subconjunto de instrucciones de secuenciamiento condicional, el valor de las señales ig, me y meu para que se cumpla la condición evaluada.

Nemotécnico	Descripción	Condición evaluada	ig	me	meu
beq	modificación del secuenciamiento si el contenido es el mismo	$rs1^v = rs2^v$	1	0	0
blt	modificación del secuenciamiento si el contenido es menor que (enteros)	$(rs1^v < rs2^v)\|_e$	0	1	0
bltu	modificación del secuenciamiento si el contenido es menor que (naturales)	$(rs1^v < rs2^v)\|_n$	0	0	1

Figura 4.33 *Condiciones evaluadas en instrucciones de secuenciamiento condicional y valores de las señales de salida del circuito EVAL. |e: comparación de enteros. |n: comparación de naturales.*

Secuenciamiento incondicional relativo y almacenamiento de la dirección de la siguiente instrucción. La instrucción "jal" determina un secuenciamiento incondicional. La dirección destino se calcula como la dirección de la instrucción de secuenciamiento más un desplazamiento. Este desplazamiento es el literal, especificado en la instrucción, formateado y multiplicado por dos (Figura 4.34). Además se almacena la dirección de la siguiente instrucción (dirección de la instrucción de secuenciamiento más cuatro) en el registro especificado del banco de registros.

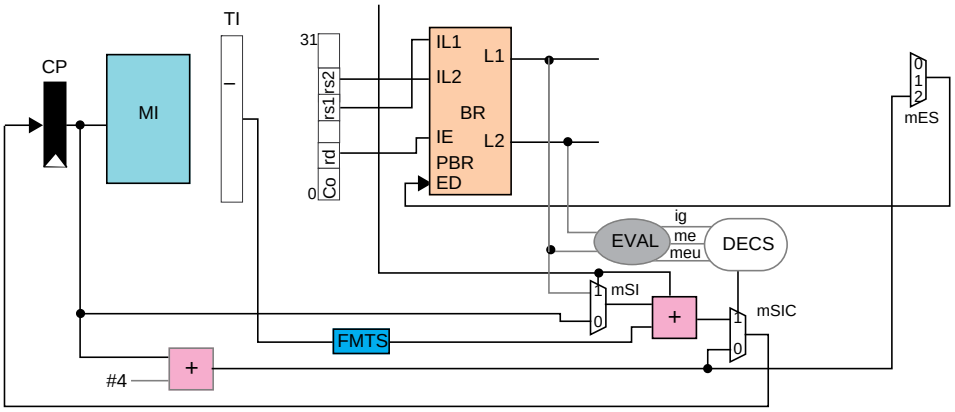


Figura 4.34 *Instrucción de secuenciamiento incondicional relativo jal.*

Secuenciamiento incondicional relativo. Un secuenciamiento incondicional relativo a la dirección de la instrucción de secuenciamiento se implementa mediante la instrucción "jal" siendo el registro destino (rd) el registro x0.

Direccionamiento indexado. La instrucción "jalr" utiliza el campo rs1 como identificador de registro fuente, el cual es entrada del puerto IL1. Al contenido del registro se le suma el literal formateado. El bit menos significativo de la suma se descarta y se establece el valor cero (Figura 4.35), lo cual requiere una señal de control en el sumador posterior al multiplexor mSI. Además se almacena la dirección de la siguiente instrucción (dirección de la instrucción de secuenciamiento más cuatro) en el registro especificado del banco de registros.

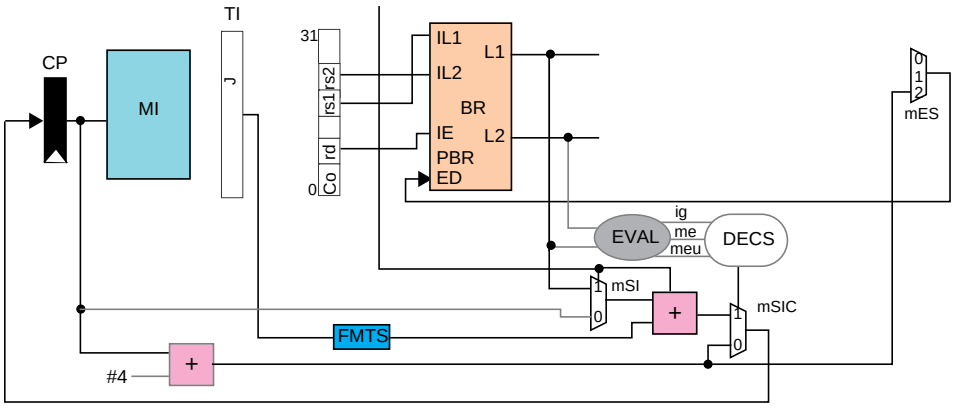


Figura 4.35 *Instrucción de secuenciamiento incondicional jalr.*

Señales de encaminamiento y actualización del banco de registros. En la Figura 4.36 se muestran las señales de control de los multiplexores que genera el autómata de control, de la señal de formateo del campo inmediato y de la señal de control del sumador posterior al multiplexor mSI. El circuito DECS está incluido en el autómata de control y las señales de entrada se muestran en la tabla de la Figura 4.36.

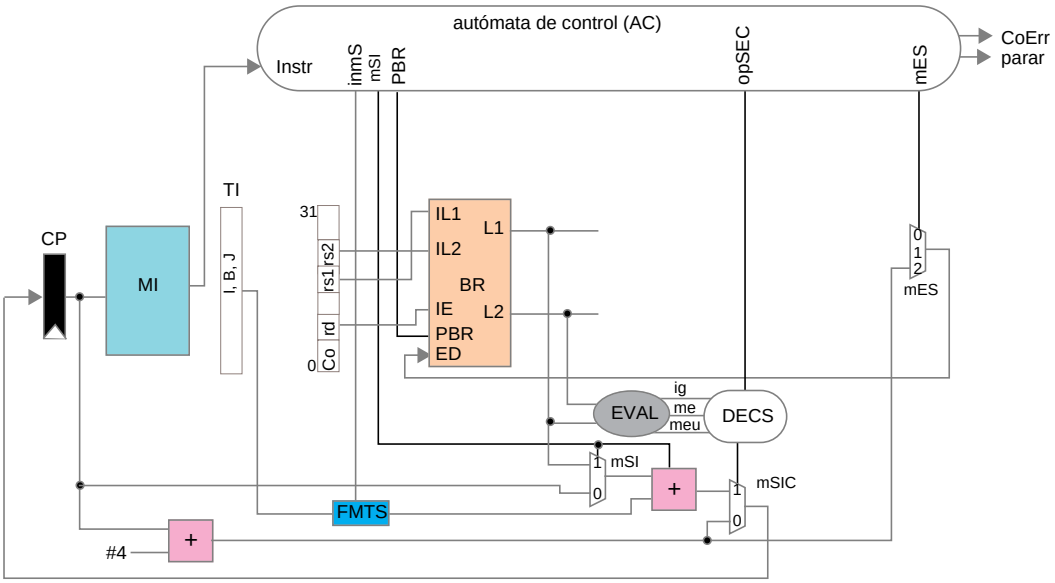


Figura 4.36 *Señales de encaminamiento de la unidad de secuenciamiento.*

En la tabla de la Figura 4.37 se muestra el valor de la señales para controlar el encaminamiento en la unidad de secuenciamiento[21]. Para las instrucciones de secuenciamiento condicional se muestran dos filas en función del resultado al evaluar la condición.

Nemotécnico	opSEC		inmS	CE	mSI	PBR	mSIC	mES
beq	SI	secuCond	SB	F	rel	no	secuImpl	
				C	rel	no	secuMod	
blt	SI	secuCond	SB	F	rel	no	secuImpl	
				F	rel	no	secuMod	
bltu	SI	secuCond	SB	C	rel	no	secuImpl	
				F	rel	no	secuMod	
jal	SI	secuInCond	UJ		rel	si	secuMod	ret
jalr	SI	secuInCond	I		indx	si	secuMod	ret

Figura 4.37 *En la columna CE se indica si la condición evaluada es cierta (C) o falsa (F).*

21. La codificación binaria de los acrónimos utilizados se especifica en el Apéndice 4.8

Preguntas

1 Traduzca las siguientes instrucciones RISC-V a lenguaje ensamblador: Un campo literal debe expresarse como entero en decimal, . . .

2 Un tipo numérico de datos en un lenguaje de alto nivel tiene un rango de valores limitado. En consecuencia, al efectuar operaciones algebraicas es . . .

3 Suponga que el procesador interpreta una instrucción de secuenciamiento condicional que cumple la condición. Indique las instrucciones de secuenciamiento que pueden generar los valores de salida del módulo EVAL . . .

Quinto nivel: unidad aritmético lógica (ALU)

En la Figura 4.38 se muestra un esquema de la ALU. De izquierda a derecha se identifican las siguientes unidades funcionales: a) desplazamiento lógico o aritmético a la derecha o a la izquierda (como máximo 31 posiciones), b) operación lógica, y c) sumador algebraico y comparación menor (enteros o naturales). El módulo slt/sltu formatea la salida de condición del sumador.

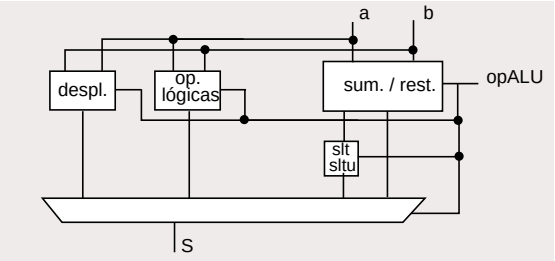


Figura 4.38 *Unidades funcionales disponibles en la ALU.*

En la tabla de la Figura 4.99 (Apéndice 4.6) se especifica el valor de las señales de control de la ALU, las cuales se utilizan para seleccionar la entrada del multiplexor de salida de la unidad mostrado en la Figura 4.38.

Sexto nivel: unidades funcionales de la ALU

En este apartado se describen parcialmente dos unidades funcionales incluidas en la ALU (Figura 4.38): el módulo sumador y el módulo desplazador (lógico y aritmético).

Suma algebraica. En la Figura 4.39 se muestra un circuito que permite realizar sumas y restas. Se utiliza un sumador binario de n bits y un conjunto de puertas “xor”.

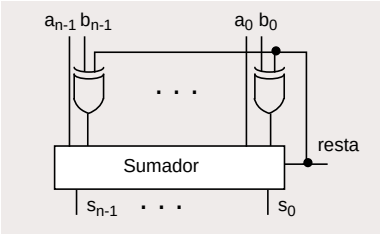


Figura 4.39 *Sumador y restador en complemento a dos.*

Otras operaciones básicas que se realizan en la ALU son las denominadas desplazamientos. Un desplazamiento de un vector de bits puede ser fijo o variable. Efectuar un desplazamiento fijo en hardware es trivial, ya que sólo es necesario encaminar directamente los cables que transportan los bits. Por tanto, nos centraremos en el caso de un desplazamiento variable.

Distinguiremos tres tipos de desplazamiento y cada uno de ellos puede ser a la izquierda o a la derecha.

Rotación. Rotación de los bits en círculo. Las posiciones que quedan vacías se rellenan con los bits que se descartan por el otro lado.

Desplazamiento lógico. Desplazamiento de los bits a la derecha o a la izquierda y las posiciones que quedan vacías se rellenan de ceros.

Desplazamiento aritmético. Igual que el desplazamiento lógico, pero cuando el desplazamiento es a la derecha, las posiciones más significativas del vector de bits que quedan vacías se rellenan replicando el bit de signo, ya que el vector de bits se interpreta en complemento a dos.

Un desplazamiento aritmético es una operación de multiplicación o división por una potencia de dos en función de si el desplazamiento es a la derecha o a la izquierda.

Dado un vector de n bits, conceptualmente una rotación requiere un conjunto de n multiplexores de n entradas, para seleccionar cada una de las salidas a partir de cada una de las entradas. Los desplazamiento lógicos y aritméticos son similares a rotaciones pero rellenando las posiciones descartadas que se crean con ceros o con unos.

En la parte izquierda de la Figura 4.40, para un vector de cuatro bits, se muestra un circuito que efectúa rotaciones a la derecha y ha sido implementado mediante multiplexores de dos entradas. Con la primera fila de multiplexores se puede efectuar una rotación de dos unidades, mientras que con la segunda fila la posible rotación es de una unidad. El control de los multiplexores se efectúa mediante el vector de bits K (k_1, k_0).

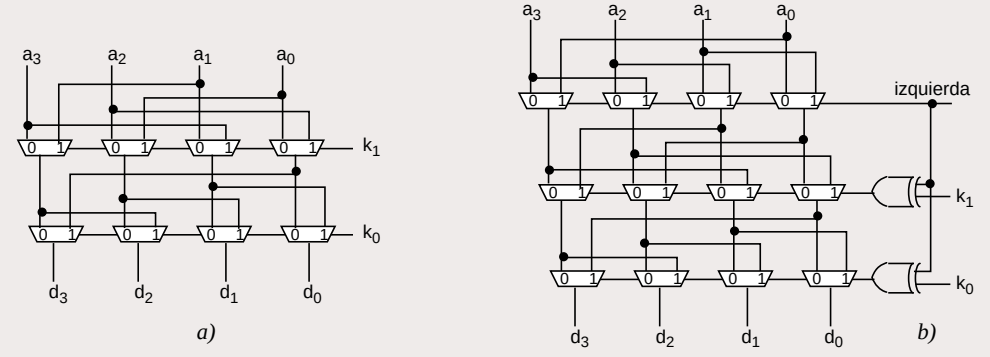


Figura 4.40 *a) Circuito que efectúa rotaciones a la derecha (entre 0 y 3 posiciones), b) circuito que efectúa rotaciones a la derecha e izquierda.*

En la parte derecha de la Figura 4.40 se muestra una modificación del circuito que permite realizar también rotaciones a la izquierda. Se añade una fila de multiplexores que permite realizar una rotación a la derecha de una posición y se modifica el control de las siguientes dos filas respecto de una rotación a la derecha. Esto es, cuando la rotación es a la izquierda se utiliza la primera fila de multiplexores y el vector K, de control de los multiplexores, se complementa.

El circuito de la Figura 4.40 se puede utilizar para construir un módulo que permita realizar desplazamientos lógicos y aritméticos. Para ello se añade en su salida un circuito que se denomina “Máscara y sustitución”. Su función es introducir ceros o unos en las posiciones que se rellenarían con los bits descartados al efectuar una rotación. Para ello se construye una máscara a partir del vector K. La tabla de verdad de la máscara se muestra en una tabla en el centro de la Figura 4.41. Posteriormente se utiliza la máscara y las restantes señales de entrada del circuito denominado “Máscara y sustitución” para determinar las posiciones donde se establecen ceros o unos. En la parte derecha de la Figura 4.41 se muestra parte del circuito. En concreto se muestra, para un bit, el circuito que permite obtener un desplazamiento a la derecha a partir de una rotación a la derecha. El bit que ha sido rotado se sustituye por un valor obtenido a partir de la máscara.

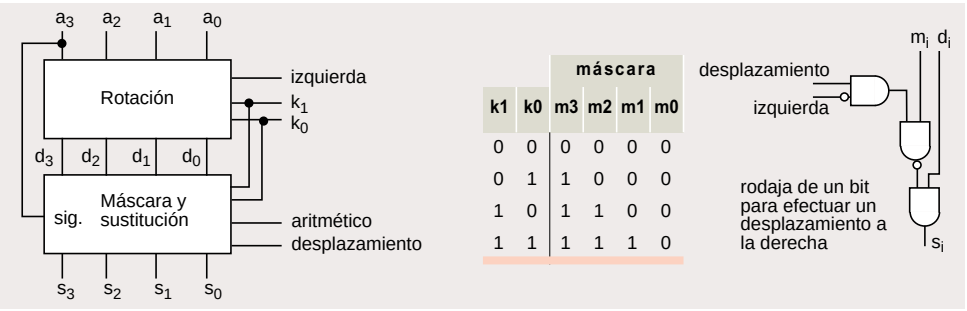


Figura 4.41 *Módulo que realiza rotaciones y desplazamiento lógicos y aritméticos.*

Nota: Para esta práctica ha sido suministrado un fichero que al desempaquetarlo crea una estructura de directorios, incluyendo alguno de ellos ficheros. El directorio raíz es LAB4, el cual incluye tres directorios: PROC_SERIE, PROC_SERIE_alu y PROC_SERIE_fmte. Cada uno de ellos incluye un árbol de directorios.
El directorio PROC_SERIE contiene el diseño base del procesador y de las memorias. El árbol de directorios correspondiente se describe en Apéndice 4.9. En este directorio está disponible un subdirectorio denominado "documentació" donde se ubica la documentación generada con Doxygen. En el Apéndice 4.12 se indica la forma de acceder a la documentación.
Los otros directorios contienen propuestas de modificaciones que se deben diseñar. En el directorio PROC_SERIE_alu se propone un proyecto que refina el diseño de la ALU. En PROC_SERIE_fmte se rediseña el formateador de datos al escribir en memoria.
Para cada diseño alternativo, el árbol de directorios es mimético al árbol del diseño base, incluyendo solo los subdirectorios que contienen ficheros modificados respecto al diseño base.

Trabajo: En el subdirectorio PROC_SERIE_alu/PROCESADOR/CAMINO_DATOS/UCalculo/COMPONENTES/ALU/ están ubicados los ficheros asociados al diseño de la ALU (Figura 4.38).

Directorio
PROC_SERIE_alu/PROCESADOR/CAMINO_DATOS/UCalculo/COMPONENTES/ALU/
CODIGO
componentes_ALU_pkg.vhd,
despla.vhd, logica.vhd, mx2.vhd, slt.vhd
alu.vhd
sumalg.vhd
QUARTUS
PRUEBAS

El fichero alu.vhd contiene la descripción estructural de la ALU. Las unidades funcionales

de desplazamiento, operaciones lógicas y el formateador ya están diseñados. En este proyecto se debe completar el diseño del sumador algebraico (fichero sumalg.vhd) y añadir la selección de las salidas de las unidades funcionales en la descripción estructural.
Compruebe el diseño efectuado. Para ello dispone de los ficheros oportunos en los directorios QUARTUS y PRUEBAS. El programa de prueba compara, para un conjunto reducido de vectores, las salidas de la alu diseñada y la alu original.

Preguntas

4 En la figura se muestran las 3 unidades funcionales de la ALU (page 250) y el árbol de multiplexores de selección: a) desplazamiento lógico o . . .

Segundo nivel: organización de la memoria de datos

La memoria de datos está organizada en cuatro bancos (Figura 4.42). Cada posición de un banco almacena un byte y las posiciones de memoria están entrelazadas entre los bancos utilizando los dos bits menos significativos de una dirección. Las entradas SCx en los bancos se utilizan para seleccionar los bancos accedidos en una operación de acceso a memoria.

En la especificación de lenguaje máquina, en un acceso a memoria se pueden leer o escribir tres tamaños de datos: a) byte, b) dos bytes, c) cuatro bytes o palabra. Para soportar los distintos tamaños de acceso se utilizan dos circuitos formateadores (Figura 4.43). Uno de ellos se ubica antes de los bancos de memoria (FMTLE) y es utilizado al interpretar las instrucciones de almacenamiento en la memoria de datos. El otro circuito formateador (FMTL) se ubica después de los bancos de memoria y es utilizado al interpretar las instrucciones de lectura de la memoria de datos.

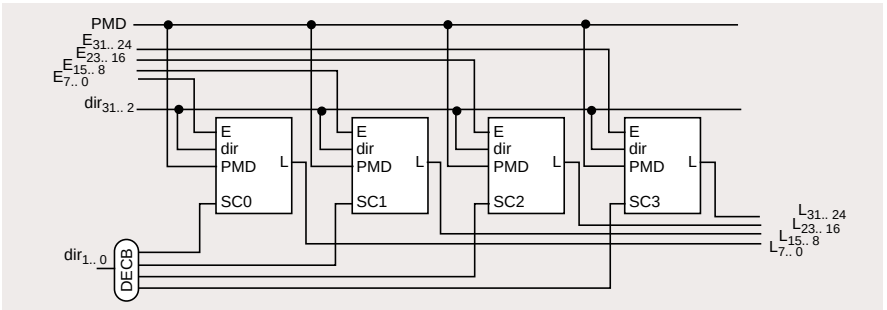


Figura 4.42 *Organización de la memoria de datos. PMD es la señal de permiso de escritura. El símbolo $|_{a...b}$ indica los bits que se consideran en un vector de 32 bits.*

El circuito acceso_MD se utiliza para comprobar si el acceso es alineado y si está dentro del rango permitido de direcciones (MD_NA, MD_FR)[22]. En cualquiera de los dos casos, la activación de la señal correspondiente es utilizada por el autómata de control para que actúe en consonancia.

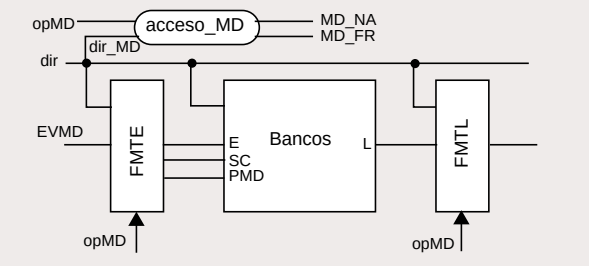


Figura 4.43 *Esquema de la memoria de datos con los circuitos formateadores. La activación de la señal MD_NA indica dirección no alineada y la activación de la señal MD_FR indica que la dirección está fuera del rango permitido.*

El circuito FMTE alinea el dato proveniente del banco de registros para actualizar los bancos de memoria correspondientes en una operación de escritura. Para ello se utilizan las señales de control opMD y los dos bits de menor ponderación de la dirección efectiva. Cuando la dirección no está alineada la salida del circuito FMTE es indeterminada. En la Figura 4.44 se muestran ejemplos de funcionamiento del circuito FMTE. En el caso a) el acceso es a byte y los dos bits menos significativos de la dirección son “01”. En el caso b) el acceso es a media palabra y los dos bits menos significativos de la dirección son “10”. En el caso c) el acceso es a palabra y los dos bits menos significativos de la dirección son “00”.

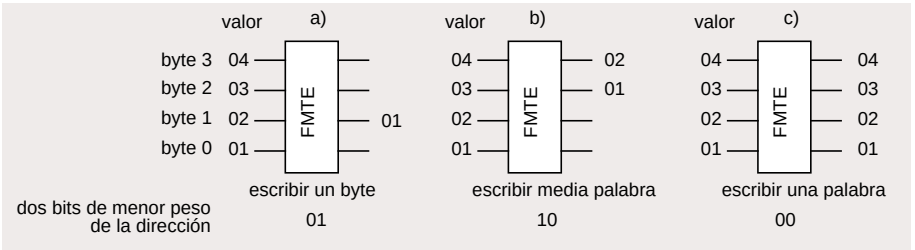


Figura 4.44 *Ejemplos de funcionamiento del módulo FMTE.*

La función del circuito FMTL es posicionar los datos leídos en los bits menos significativos de la salida y rellenar el resto de bits más significativos, hasta 32 bits, con ceros o unos en función del código de operación y del dato leído de la memoria de datos. En la Figura 4.27 se ha mostrado la codificación de la señal de control de la memoria de datos.

22. El tamaño utilizado de la memoria de datos es 4KBytes.

En la Figura 4.45 se muestran ejemplos de funcionamiento del circuito FMTL. La salida de este circuito está indeterminada cuando la dirección no está alineada. En el caso a) la lectura es de un byte y los dos bits menos significativos de la dirección son “01”. En el caso b) la lectura es de media palabra y los dos bits menos significativos de la dirección son “10”. En el caso c) la lectura es de una palabra. En el caso b) se muestran dos columnas de valores en la salida del circuito. La primera columna de valores se corresponde con una lectura que se interpreta como un valor entero (complemento a dos) y la segunda columna se interpreta como un valor natural.

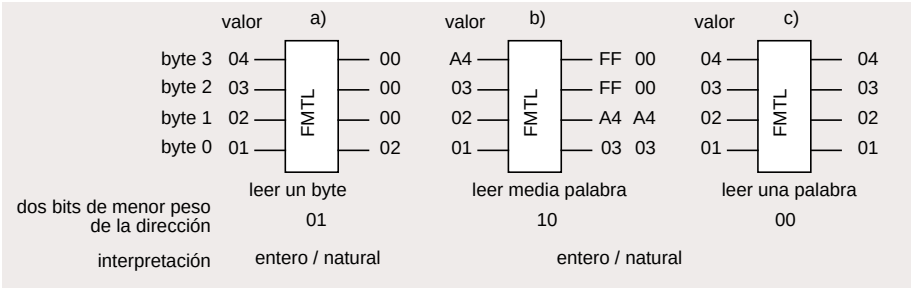


Figura 4.45 *Ejemplos de funcionamiento del módulo FMTL.*

Preguntas

En las siguientes dos preguntas se utiliza el diseño ubicado en el directorio PROC_SERIE. El Apéndice 4.9 muestra la organización del proyecto en directorios.

5 El circuito FMTL (fichero FMTL.vhd) formatea el dato leído de la memoria que se utiliza para actualizar el banco de registros . . .

6 El circuito FMTE formatea el dato leído del banco de registros para actualizar los bancos de la memoria de datos . . .

Trabajo: En el subdirectorio PROC_SERIE_fmte/MD/COMPONENTES/FMTE/COMPONENTES/ALINEARE/ están ubicados los ficheros asociados a un diseño alternativo del formateador FMTE (Figura 4.43).

Directorio
PROC_SERIE_fmte/MD/COMPONENTES/FMTE/COMPONENTES/ALINEARE/
CODIGO
alinearE.vhd
QUARTUS
PRUEBAS
prueba_alinearE.vhd

En este diseño se utilizan los 2 bits menos significativos de la señal de control opMD, que indican la granularidad del acceso, para formatear el dato de escritura a memoria. Compruebe el diseño efectuado. Para ello dispone de los ficheros oportunos en los directorios QUARTUS y PRUEBAS. Modifique el programa de prueba para comprobar el diseño.

Preguntas

En la siguiente pregunta se utiliza el proyecto ubicado en el directorio PROC_SERIE_fmte. No es necesario utilizar el constructor “generate”.

7 Proponga un diseño alternativo del módulo alinearE que utilice únicamente los 2 bits menos significativos de la señal de control opMD . . .

Segundo nivel: memoria de instrucciones

Las instrucciones se almacenan en posiciones de memoria alineadas a cuatro bytes. En consecuencia, los dos bits menos significativos de una dirección, utilizada para leer una instrucción, son siempre cero. Si este no es el caso se genera una señal de excepción (MI_NA, parte derecha de la Figura 4.46).

La memoria de instrucciones está organizada como la memoria de datos. Una instrucción ocupa 4 bytes. La dirección que se utiliza para acceder a la memoria de instrucción es el valor almacenado en el registro CP (parte izquierda de la Figura 4.46).

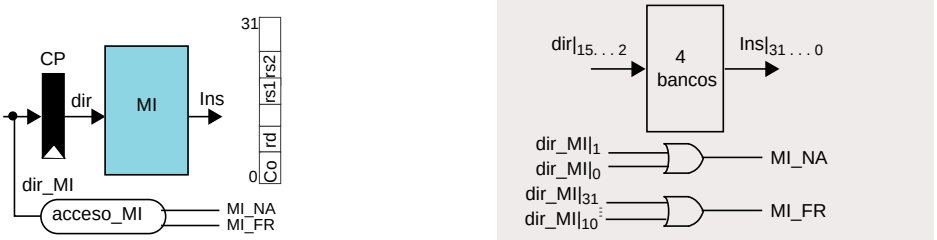


Figura 4.46 *Memoria de instrucciones.*

El tamaño disponible en la memoria de instrucciones es 2^{10} bytes y el número de instrucciones que se puede almacenar es 2^{8}. Esto es, los 22 bits más significativos de la dirección en la entrada del registro CP no se utilizan. Si estos bits son distintos de cero se genera una señal de excepción (MI_FR, Figura 4.46). La activación de las señales MI_NA y MI_FR determina que el autómata de control también active la señal parar.

Camino de datos completo

En la Figura 4.47 se muestra el esquema completo del camino de datos del procesador, en el cual se han especificado todas las señales menos la señal de reloj y la señal de inicialización.

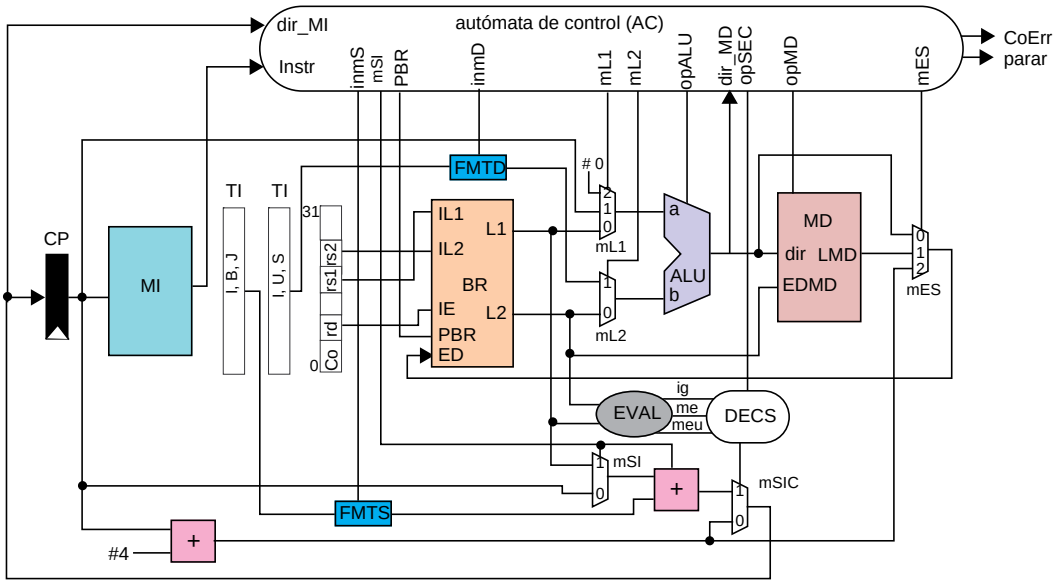


Figura 4.47 *Camino de datos del procesador.*

Camino de datos: interconexión entre entidades

En la Figura 4.48 se muestra la interconexión de los distintos componentes del procesador. Solo se muestra el flujo de información entre componentes. En el autómata de control se desagrega la parte correspondiente a la detección y gestión de condiciones de error. En particular, la gestión es parar la interpretación instrucciones.

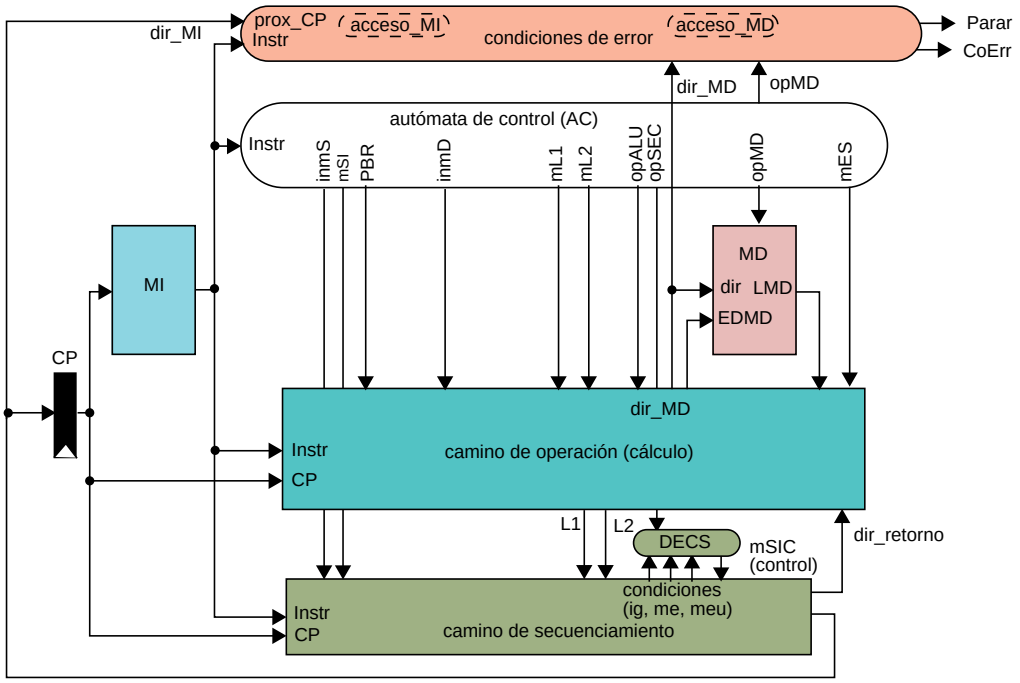


Figura 4.48 *Interconexión de los componentes del procesador.*

Cuarto nivel: autómata de control

El autómata de control se encarga de decodificar las instrucciones y generar las señales de control del camino de datos y de las unidades funcionales. Las entradas del autómata son una instrucción y señales de condición que se evalúan en el camino de datos.

En la Figura 4.49 se muestra la estructura del autómata de control. En ella se distingue un módulo denominado DEC que determina las señales de encaminamiento de la unidad de cálculo (encaminamiento_UC), las señales de control de la memoria de datos (opMD), señales de secuenciamiento condicional e incondicional (opSEC) y las señales para control de la ALU (opALU). La señal de control del secuenciamiento (encaminamiento_SEC) se obtienen a partir de opSEC y el resultado de la evaluación de la condición (EVAL), si es el caso.

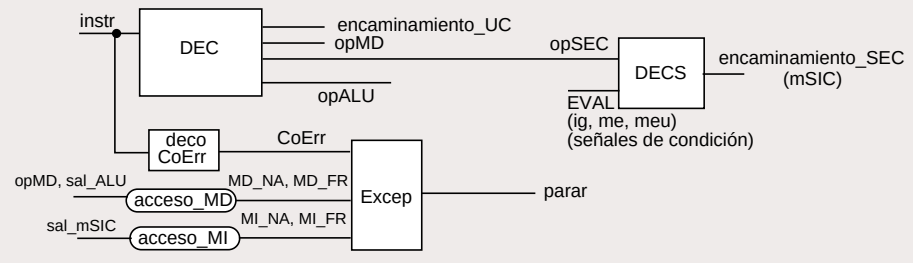


Figura 4.49 *Módulos en el autómata de control. DEC: decodificación y gestión de las señales de error. DECS: segundo nivel de decodificación y control para las instrucciones de secuenciamiento y gestión de las señales de condición. decoCoerr: identificación de secuencias de bits que no pertenecen al lenguaje máquina implementado.*

Asociado al decodificador está la detección de condiciones de excepción (decoCoErr, acceso_MD, acceso_MI). Las señales de error se validan en los módulos que las generan. Por ejemplo, la señal MD_NA no se activa si la instrucción no es de acceso a memoria.

El gestor de condiciones de error (Excep) al observar la activación de una de las señales de error activa la señal parar. Esta última señal también se activa al decodificar instrucciones que no pertenecen al subconjunto del lenguaje máquina implementado.

Quinto nivel: organización del decodificador

El decodificador (DEC) se divide en varios decodificadores o partes para considerar los casos en los cuales es necesario decodificar campos adicionales al campo código de operación. En concreto, los códigos de operación LOAD, STORE, OP_IMM, OP y BRANCH requieren decodificar otros campos de la instrucción (Figura 4.50).

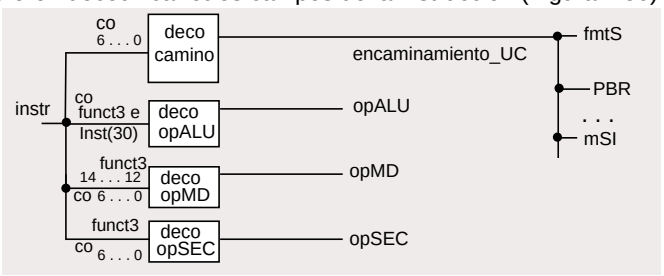


Figura 4.50 *Organización del decodificador DEC. El módulo decocamino utiliza los bits del código de operación de una instrucción para generar las señales de encaminamiento. Los otros módulos utilizan los bits mostrados para generar las señales decontrol de las unidades funcionales. Los dos bits menos significativos del campo "co" siempre tienen el valor 1 para tamaños de instrucciones mayores de 16 bits.*

La señal de permiso de escritura en el banco de registro está incluida en el conjunto de señales denominadas encaminamiento.

La salida del decodificador (DEC) para controlar las unidades funcionales son: opALU, opMD, opSEC. El bit más significativo del conjunto de bits de cada señal de control indica si se utiliza la unidad funcional[23] para ejecutar la instrucción.

Espacio lógico

El código de un programa empieza en la dirección x"00000100" y las instrucciones se almacenan en posiciones contiguas de memoria a partir de dicha dirección.

En las direcciones previas del espacio lógico se almacenan instrucciones que: a) inicializan el registro que contiene la dirección que indica la cabeza de la pila, y b) instrucciones que establecen como valor del registro CP la primera dirección del programa. A esta secuencia de instrucciones la denominamos código de inicio (Figura 4.51).

Después de código de inicio hay una secuencia de instrucciones, denominada código de finalización, que suspende la interpretación de instrucciones. La última instrucción de un programa establece como valor del registro CP la primera dirección del código de finalización.

direc.	leng. maq.	Instrucciones de RISC V en ensamblador		Código de inicialización y finalización	Comentarios
				# codigo de inicialización	
				# sp: se utiliza el registro x2	
text					Código de inicio
entry:					Establece el valor de sp
0:	10001117	auipc	x2, 0x10001	la sp,__sp	sp: x2
4:	00010113	addi	x2, x2, 0 # 10001000 <__mem_top__>	addi sp,sp,-32	Reserva 32 bytes en la pila
8:	fe010113	addi	x2, x2, -32	add x1, x0, x0	Inicialización de registros
c:	000000b3	add	x1, x0, x0	add x3, x0, x0	A partir de x3 en secuencia
10:	000001b3	add	x3, x0, x0	add x4, x0, x0	
14:	00000233	addi	x4, x0, x0	. . .	
. . .		. . .		add x30, x0, x0	
80:	00000fb3	addi	x31, x0, x0	add x31, x0, x0	
84:	07c000ef	jal	x1, 100 <main>	call main	
88:	00000013	addi	x0, x0, # 0	nop	
. . .		. . .		. . . (4 nop)	Código de finalización
9c:	00100073	ebreak		ebreak	
100:			<main>		Código del programa

Figura 4.51 *Códigos de inicio y finalización. La primera columna de la izquierda indica los bits significativos de la dirección en hexadecimal. La segunda columna muestra la instrucción en lenguaje máquina. La tercera columna muestra instrucciones RISC V en ensamblador. La siguiente columna muestra instrucciones en pseudoendamblador.*

23. En este contexto identificamos el secuenciamiento explícito como unidad funcional.

Los datos globales se almacenan a partir de la dirección x"10000000". La primera posición de la pila es la dirección x"10000FFF" y su tamaño se incrementa hacia direcciones decrecientes.

El código de un programa y los códigos de inicio y finalización se almacenan en la memoria de instrucciones (MI, Figura 4.52). Los datos globales y la pila se almacenan en la memoria de datos (MD).

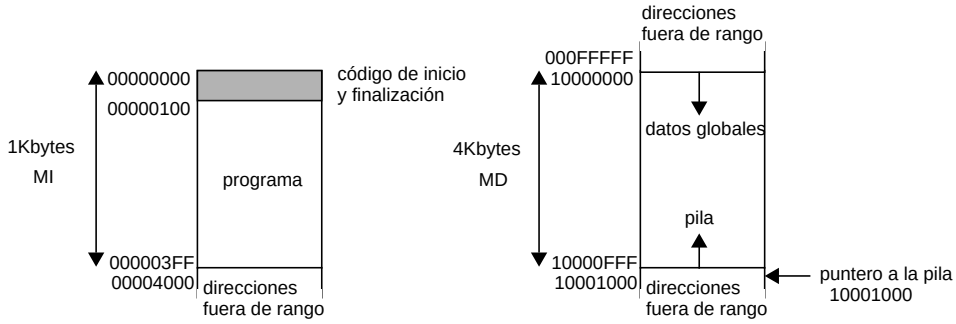


Figura 4.52 *Espacio lógico y asociación a las memorias del procesador.*

Señal de reloj, señal de inicio y elementos de almacenamiento

El registro CP se actualiza en el flanco ascendente de la señal de reloj (Figura 4.47 y Figura 4.53). Al activar la señal inicializar (Figura 4.10, señal PCERO en el fichero de simulación) el registro CP toma el valor cero. En consecuencia, se lee de la memoria de instrucciones (MI) la instrucción que está en la dirección cero.

El banco de registros se actualiza en el flanco ascendente de la señal de reloj si el permiso de escritura (PBR) está activado (Figura 4.53).

Un banco de la memoria de datos se actualiza en el flanco ascendente de la señal de reloj (Figura 4.53) cuando el permiso de escritura está activado (PMD, Figura 4.42) y el banco ha sido seleccionado (SCx, Figura 4.42).

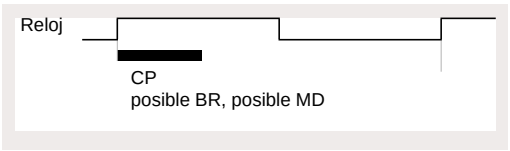


Figura 4.53 *Señal de reloj y actualización de los elementos de memorización. El retardo de actualización no está a escala.*

Simulación

Los ficheros que utiliza Quartus (proc_MD_MI.qsf y proc_MD_MI.qpf) se encuentran en el directorio LAB4/PROC_SERI/ENSAMBLADO/QUARTUS.

Una vez ha sido abierto el proyecto[24] con Quartus, hay que dar la orden “Processing -> Start -> Start Analysis & Elaboration” . Esta orden elabora el diseño. Una vez elaborado el diseño sin errores, mediante la orden “Tools -> Netlist Viewers -> RTL Viewer” se puede comprobar la elaboración efectuada (Apéndice 4.10).

El fichero proc_MD_MI.vhd contiene la descripción en VHDL del procesador base. Este fichero está ubicado en el directorio LAB4/PROC_SERIE/ENSAMBLADO/CODIGO[25] [26]. El esquema RTL, elaborado por Quartus, se muestra en la Figura 4.54.

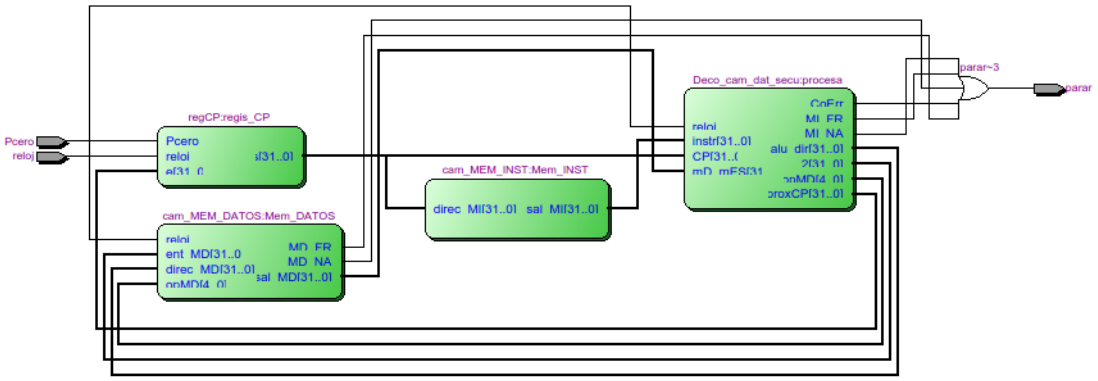


Figura 4.54 *Esquema RTL del procesador.*

Preparación de la simulación

El fichero de comprobación del diseño base, denominado prueba_proc_MD_MI.vhd se encuentra en el directorio LAB4/PROC_SERIE/ENSAMBLADO/PRUEBAS. En este directorio también se encuentra el fichero wave.do para formatear las señales que se visualizan en la ventana temporal.

En el fichero prueba_proc_MD_MI.vhd se especifica un proceso para generar la señal de reloj, la cual se utiliza en la simulación.

24. En los diseños alternativos también están disponibles los respectivos proyectos para su elaboración y simulación.
25. La organización del proyecto en directorios se muestra en el Apéndice 4.9.
26. En el Apéndice 4.12 se indica una posibilidad para analizar el diseño y su estructura utilizando un navegador.

El periodo de la señal de reloj se especifica como un genérico (periodo_reloj). En la declaración de la entidad se utilizan otros genéricos (Figura 4.55). Mediante ellos se controla la simulación.

Genérico	Descripción (activo: valor true)	Valor
periodo_reloj	periodo de la señal de reloj	80 ns
pasoapaso	permite una simulación ciclo a ciclo	true
impimir_traza	imprime información de las instrucciones a medida que se interpretan	true
imprimir_MD	imprime el contenido de la memoria de datos al finaliza la simulación	true
imprimir_MI	imprime el contenido de la memoria de instrucciones al finaliza la simulación	true
imprimir_BR	imprime el contenido del banco de registros al finaliza la simulación	true

Figura 4.55 *Parámetros de la simulación.*

Para modificar los valores por defecto utilizados en los genéricos hay que editar el fichero del programa de prueba.

Simulación con ModelSim

Para activar la simulación de Modelsim desde Quartus dé la orden que se muestra en la Figura 4.56 (RTL Simulation).

Figura 4.56 *Paleta de iconos en Quartus. Activación de simulador Modelsim desde Quartus.*

Simulación paso a paso (ciclo a ciclo). Una vez activada la simulación utilice la orden "Run-All" para avanzar un ciclo de simulación (Figura 4.57).

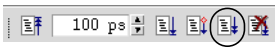


Figura 4.57 *Paleta de iconos en ModelSim. Orden Run-All*

Marcas verticales en la ventana de tiempo. Para establecer líneas verticales en cada inicio de ciclo de la señal de reloj deben efectuarse los siguientes pasos. Seleccione la ventana temporal. Posteriormente de la orden "Wave -> Wave Preferences ...". En la ventana emergentes (Figura 4.58), pestaña Grid&Time", establezca como valor en "Grid Offset" el valor, en nanosegundos, del semiperiodo de la señal de reloj (periodo_reloj/2) y en "Auto Period" el valor del periodo de la señal de reloj (periodo_reloj, Figura 4.58).

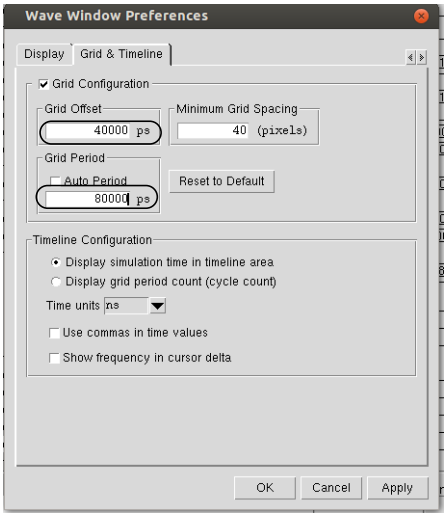


Figura 4.58 *Pasos para establecer marcas en la ventana temporal. En el ejemplo el periodo de la señal de reloj es 80 ns.*

Una vez establecidas las marcas en la ventana temporal se pueden almacenar para simulaciones posteriores. Para ello, seleccione la ventana temporal y posteriormente de la orden “File -> Save Format”. En la ventana emergente pulse en el boton “Browse...” y muévase por la jerarquía de directorios. El fichero wave.do debe almacenarse en el subdirectorio PRUEBAS. En caso de utilizarse otro nombre debe modificarse el fichero formato_ventanas.do, que está ubicado en el mismo subdirectorio.

Simulación de un programa concreto

Un programa consta de dos ficheros de texto. Uno de ellos contiene las instrucciones en lenguaje máquina y el otro almacena los datos. El nombre de los ficheros que se utilizan en la simulación son: CODI_imem.hex y DADES_dmem.hex. Estos ficheros están ubicados en el subdirectorio PRUEBAS.

ModelSim, al iniciar la simulación, lee el contenido de los ficheros de instrucciones y de datos que se especifican e inicializa las memorias. Posteriormente empieza la simulación.

Los ficheros de los programas están ubicados en el directorio LAB4/programes. En este directorio hay un subdirectorio para cada programa.

Para simular un programa hay que copiar los ficheros de código y de datos del programa en los ficheros CODI_imem.hex y DADES_dmem.hex respectivamente, ubicados en el directorio PRUEBAS. El nombre del programa está especificado en la primera línea de cada fichero.

El fichero resultado “resultados_ejecucion_programa.txt” se almacena en el subdirectorio RESULTADOS.

Observación de información almacenada en posiciones de almacenamiento

Observación de la información almacenada en el banco de registros. Pulse en el símbolo + asociado a la izquierda del nombre (BR, Figura 4.62) y se muestra el contenido de los registros durante la simulación. En la ventana temporal de la Figura 4.59 se observa la modificación del contenido de los registros 13, 14 y 15 durante una simulación.

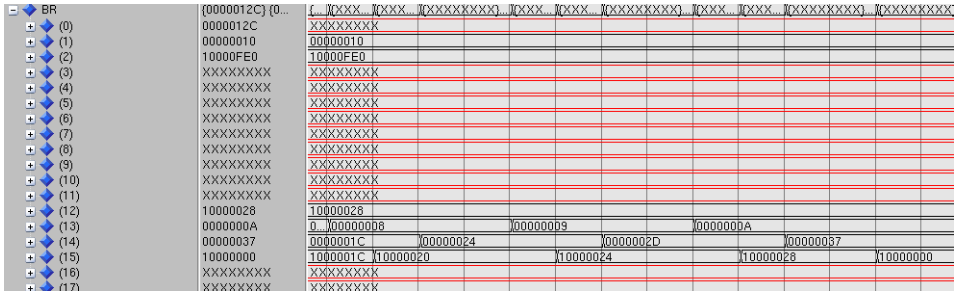
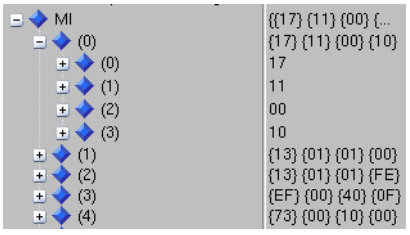


Figura 4.59 *ModelSim. Ventana temporal. Evolución del contenido de los registros del banco de registros.*

Observación de la información almacenada en MI o MD. Pulse en el símbolo + asociado a la izquierda del nombre (MI, MD, Figura 4.62) y se muestra el contenido de la memoria correspondiente durante la simulación. En una fila se muestran cuatro valores en hexadecimal. Cada valor se corresponde con un banco de la memoria correspondiente. El byte de menor ponderación está ubicado a la izquierda. Pulsando en el símbolo “+” de una fila se muestra el contenido individualizado de los 4 bancos.

En la Figura 4.60 se muestra el contenido de la MI al simular un programa. La instrucción en la dirección cero es “auipc x2, x10001” cuya codificación en hexadecimal es x”10001117”.



bits (contenido en hexadecimal)

	31:24 . . . . 7:0		
auipc r2 , x10001	10 00 11 17	0000000 00 00	dirección de palabra (instrucción)
	17	0000000 00 00	dirección de byte
	11	0000000 00 01	
	00	0000000 00 10	
	10	0000000 00 11	
addi r2 , r2 , x000	00 01 01 13		

Figura 4.60 *ModelSim: Ventana temporal: Contenido de la MI.*

En la Figura 4.61 se muestra el contenido de la MD al simular un programa. En cada fila se muestran cuatro valores en hexadecimal. Cada valor se corresponde con un banco de memoria. El byte de menor ponderación está ubicado a la izquierda. Los valores que se muestran en la segunda columna, empezando por la izquierda, se corresponden con el instante de tiempo determinado por la posición del cursor en la ventana temporal.

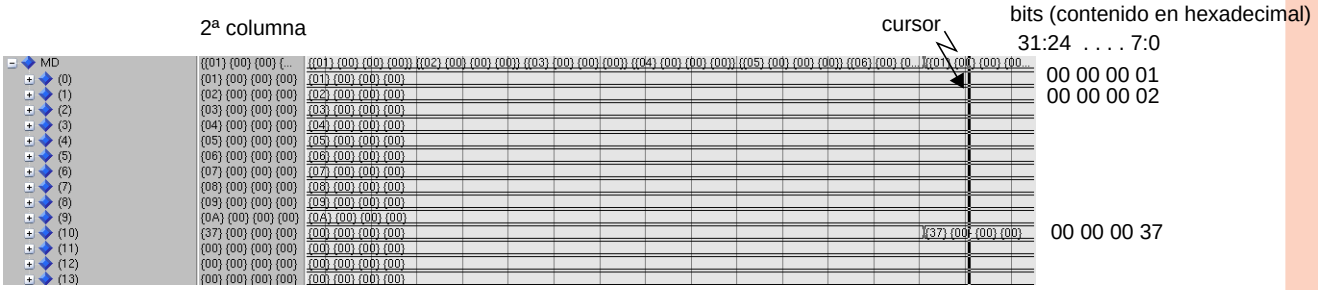


Figura 4.61 *ModelSim: Ventana temporal: Contenido de la MD.*

Evolución de las señales del camino de datos

En la Figura 4.62 se describen algunas de las señales del camino de datos que se muestran en la ventana de temporal de ModelSim. Los identificadores de registros se muestran en decimal. Las señales de encaminamiento, control de las U.F. y de evaluación de la condición se muestran en binario. El contenido de posiciones de almacenamiento y las señales “instruc” y CP se muestran en hexadecimal.

MI	Memoria de instrucciones (MI)	opALU	Controla las operaciones en la ALU
MD	Memoria de datos (MD)	opMD	Controla las operaciones en la memoria de datos (MD)
BR	Banco de registros (BR)	opSEC	Controla el secuenciamiento
me_meu_ig[a]	Salida del circuito que evalúa la condición	IDL1	Identificador de registro fuente (rs1)
sel_mSIC	Selección del multiplexor mSIC (secuenciamiento)	IDL2	Identificador de registro fuente (rs2)
mSIC	Salida del multiplexor mSIC (secuenciamiento)	IDE	Identificador de registro destino (rd)
fmts	Control del formateo en el módulo FMTS	instruc	Salida de la MI
MSI	Selección en el multiplexor mSI	CP	Salida del registro CP
PBR	Permiso de escritura en el banco de registros	ciclo	Contador de ciclos
fmtD	Control del formateo en el módulo FMTD	reloj	Reloj
mL1	Control del multiplexor mL1		
mL2	Control del multiplexor mL2		
mES	Control del multiplexor mES		

a. Las señales están ordenadas, de izquierda a derecha, en el sentido en que se muestran.

Figura 4.62 *Etiquetas utilizadas en las señales que se muestran en la ventana de tiempo.*

En la Figura 4.63 se muestra un intervalo de tiempo de la ventana temporal de ModelSim al interpretar un programa en el procesador descrito en VHDL.

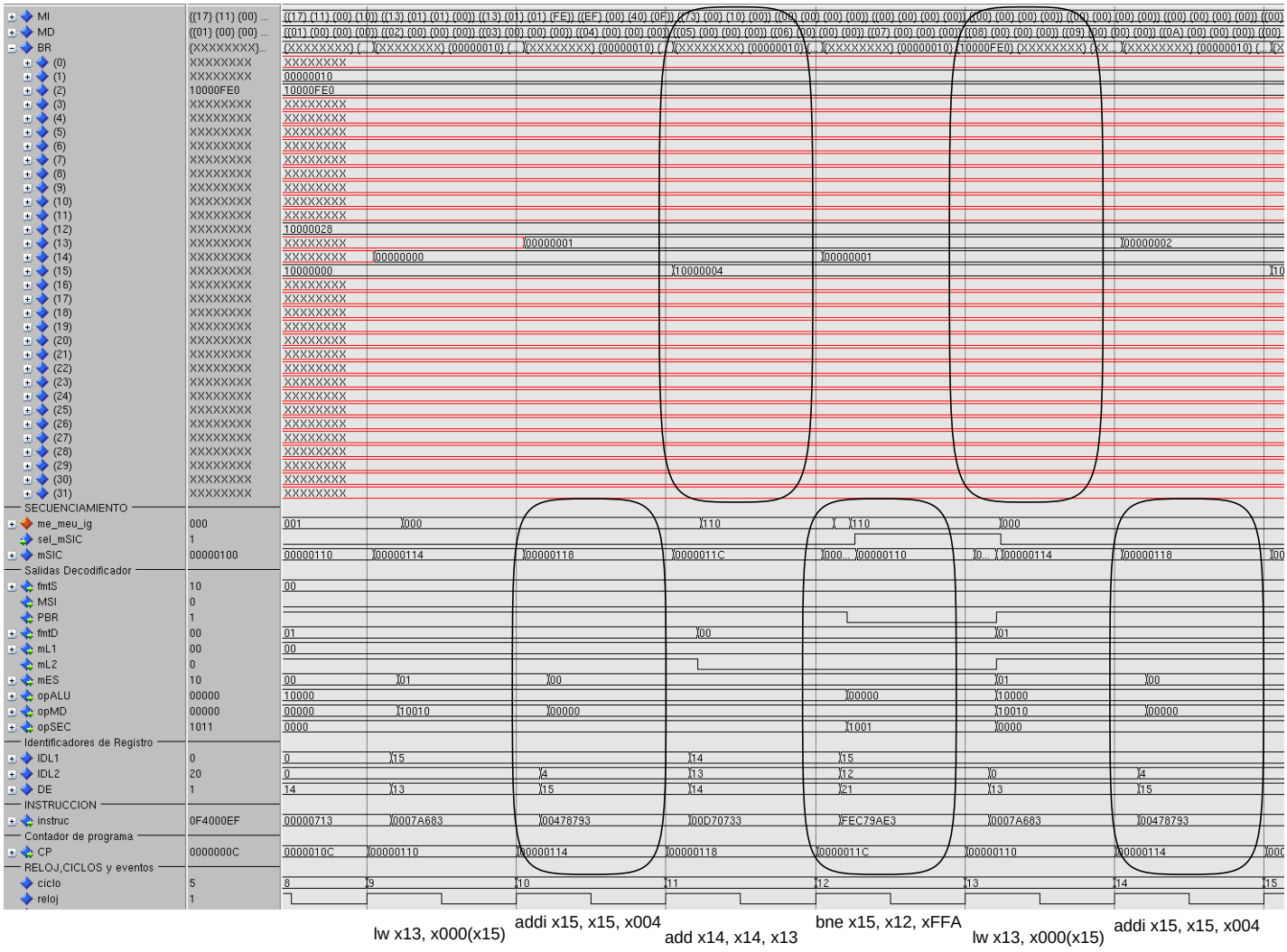


Figura 4.63 *ModelSim. Ventana temporal. Evolución de algunas señales del camino de datos.*

En el ciclo 10 se interpreta una instrucción de suma y con el resultado se actualiza el registro x15, del banco de registros, al inicio del siguiente ciclo. En el ciclo 12 se interpreta una instrucción de secuenciamiento condicional. La señal sel_mSIC se activa durante la interpretación. En el siguiente ciclo se almacena la salida del multiplexor mSIC en el registro CP.

En la Figura 4.64 se describen otras señales que se muestran en la ventana temporal de ModelSim. En primer lugar se relacionan las señales Pcero y parar. También se describen las señales que indican una condición de excepción al interpretar una instrucción.

Pcero	Señal de puesta a cero
parar	Señal de parada
CoErr	Señal de error en el código de operación o campo adicional
MD_FR_FA[a]	Señal de acceso no permitido en el acceso a la MD: dirección fuera de rango o no alineada
MI_FR_FA[b]	Señal de acceso no permitido en el acceso a la MI: dirección fuera de rango o no alineada

a. Las señales están ordenadas, de izquierda a derecha, en el sentido en que se muestran.
b. Las señales están ordenadas, de izquierda a derecha, en el sentido en que se muestran.

Figura 4.64 *Etiquetas de señales de puesta a cero y excepción en la ventana temporal.*

Información textual de la simulación

En la ventana “Transcript” de Modelsim se muestra información para cada instrucción interpretada y al finalizar la simulación se muestra el contenido del banco de registros y de las memorias.

El formato de la información relativo a la interpretación de instrucciones se indica en la Figura 4.65. En una fila se representa el ciclo en que se interpreta la instrucción, la codificación en lenguaje máquina (L.M.) y lenguaje ensamblador (L.E.). Posteriormente se muestran las señales de salida del decodificador. Seguidamente se muestran las señales de errores. Junto con ellas se muestra la salida de la ALU (dirección de acceso a memoria, si es una instrucción de acceso a memoria) y la salida del multiplexor mSIC, la cual se almacena en el registro CP en el siguiente ciclo.

ciclo	CP	Inst	Inst	fmtS	mSI	PBR	fmtD	mL1	mL2	mEs	opALU	opMD	opSEC	mSIC
número	valor en hexadecimal	instrucción en L.M. (hexadecimal)	instrucción en L.E.	binario										

CoErr	MD_FR	MD_FA	dir_MD	MI_FR	mi_FA	prox_CP
	binario		valor en hexadecimal			valor en hexadecimal

Figura 4.65 *Formato de impresión de la instrucción que se interpreta.*

El formato utilizado para mostrar el contenido de los registros del banco de registros se indica en la Figura 4.66.

número de registro	Contenido
decimal	valor en hexadecimal

Figura 4.66 *Formato de impresión del banco de registros*

El formato utilizado para mostrar el contenido de las posiciones de almacenamiento de la memoria de instrucciones se indica en la Figura 4.67. Las posiciones de almacenamiento se muestran en secuencia. Para cada instrucción se muestran 4 bytes. En una fila se muestran 4 bytes.

Posiciones de almacenamiento de 4 bytes consecutivas con el mismo valor no se representan. Se representa el primer conjunto de 4 bytes con el valor que es idéntico. Posteriormente se imprime una fila ". . . . .". Seguidamente se imprime otra fila con el valor identificado previamente, que es idéntico. Posteriormente se imprimen filas con valores distintos.

Dirección						Contenido	
bits: 31 : 4	bits 3:2		bits 1:0			L.M.	L.E.
hexadecimal	binario	11	10	01	00	hexadecimal	ascii

Figura 4.67 *Formato de impresión de la MI.*

El formato utilizado para mostrar el contenido de las posiciones de almacenamiento de la memoria de datos es idéntico al formato que se utiliza para imprimir la memoria de instrucciones, excluyendo la última columna (interpretación de los 4 bytes).

Toda la información que se muestra también se almacena en el fichero de resultados ubicado en el directorio RESULTADOS.

Tiempo de ciclo

El tiempo de ciclo está predeterminado por el camino con mayor retardo. Para determinarlo hay que tener en cuenta los elementos utilizados del camino de datos, su relación de precedencia y el retardo de cada uno de ellos. Para ello, debemos tener en cuenta que la ubicación espacial de un componente en un camino de datos no determina su precedencia.

Por otro lado, hay que tener en cuenta todos los tipos de instrucción que se interpretan. Como el camino de datos es unificado, las salidas de algunos elementos no son utilizados por algún tipo de instrucción. Por ello, hay que calcular el peor caso. Esto es, el retardo máximo que necesitan algunas instrucciones.

En la Figura 4.68 se muestra el camino de datos con los retardos asociados a los componentes. Los retardos de los componentes utilizados se detallan en el Apéndice 4.11. Estos retardos no son representativos de un diseño. Sólo son de utilidad para efectuar los cálculos de retardo que se muestran o solicitan.

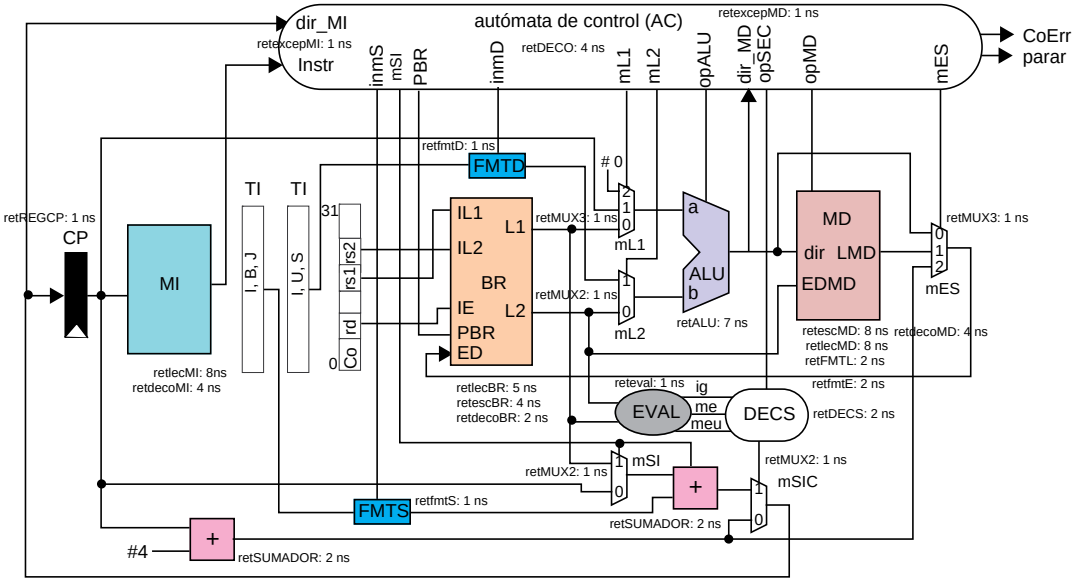


Figura 4.68 *Camino de datos y retardo de los componentes.*

Para facilitar el cálculo del tiempo de ciclo utilizaremos un cronograma donde se representa el retardo de los elementos del camino de datos. En la Figura 4.69 se muestran los acrónimos utilizados para denominar componentes del camino de datos.

retBRE	Retardo en la actualización de un registro del banco de registros	+	Sumador
decBRE	Decodificador del puerto de escritura del banco de registros. No está representado en las figuras	EVAL	Unidad de evaluación de la condición
mES	Multiplexor en la etapa ES	mSI	Multiplexor
FMTL	Módulo para formatear el dato con el cual se actualiza el banco de registros	mL2	Multiplexor para seleccionar el operando b que utiliza la etapa ALU
MDE	Retardo de escritura en memoria	mL1	Multiplexor para seleccionar el operando a que utiliza la etapa ALU
MDL	Retardo de lectura en memoria	retBRL	Retardo en la lectura de un registro del banco de registros
excepMI	Lógica de excepción	FMTS	Formateador del campo literal en la USE
excepMD	Lógica de excepción	FMTD	Formateador del campo literal en la UC
decoMD	Decodificador de la dirección de memoria	deco	Retardo del decodificador
ALU	Módulo ALU	MI	Retardo de lectura de la ,emoria de instrucciones
FMTE	Módulo para formatear el dato con el cual se actualiza memoria	decoMI	Decodificador de la dirección de memoria
mSIC	Multiplexor	+	Sumador
DECS	Módulo de secuenciamiento condicional		

Figura 4.69 *Acrónimos del retardo de los componentes del camino de datos.*

Todos los retardos deben interpretarse como el retardo del componente. Alguno de los componentes están incluidos dentro de un módulo que no se visualiza. Por ejemplo, el decodificador del identificador del registro cuando se escribe en un registro del banco de registros.

En la Figura 4.70 se muestra un cronograma. En primer lugar se tiene en cuenta el retardo del registro CP. Seguidamente, en paralelo se observa el retardo del sumador (2) y del decodificador de la memoria de instrucciones (3).

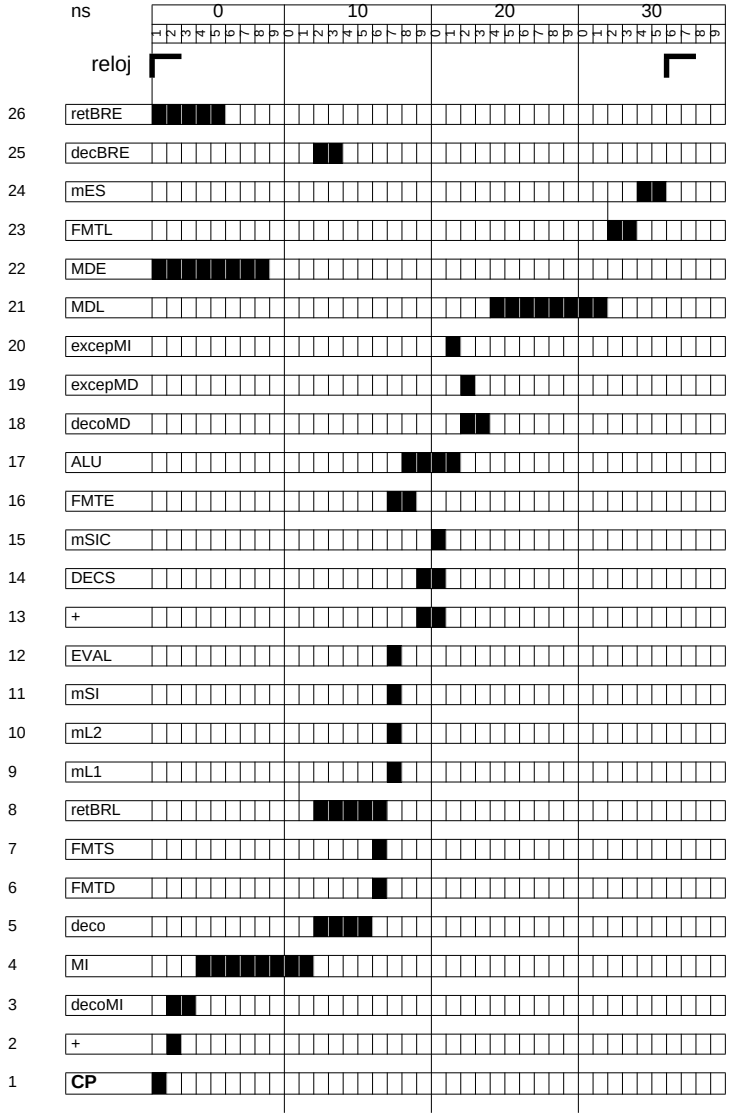


Figura 4.70 *Cronograma de retardos del camino de datos.*

El siguiente elemento que hay que considerar es MI (4). Una vez la instrucción ha sido leída de MI actúan en paralelo el decodificador de instrucciones (5), el decodificador del identificador del banco de registros del puerto de escritura (25) y la lectura de registros del BR (8).

Una vez la salida del decodificador es estable hay que tener en cuenta el retardo de FMTD (6) y FMTS (7).

Una vez la salida del decodificador es estable y las salidas de los puertos de lectura del banco de registros son estables hay que tener en cuenta el retardo de los componentes mL1 (9), mL2 (10), mSI (11), EVAL (12) y FMTE (16), los cuales disponen de un valor estable en las entradas en el mismo instante de tiempo.

Una vez las salidas de los multiplexores son estables, los elementos que evalúan en paralelo son el sumador a la salida del multiplexor mSI (13), DECS (14) y la ALU (17).

El retardo del multiplexor mSIC (15) hay que tenerlo en cuenta cuando las salidas de DECS son estables y la salida del sumador, en la salida del multiplexor mSI, es estable.

Cuando la salida de la ALU es estable hay que considerar el retardo del elemento acceso_MD (19, excep_MD). De forma similar, cuando la salida de mSIC es estable hay que considerar el retardo de acceso_MD (20, excepMI).

Una vez la salida de la ALU es estable se consideran los retardos del decodificador de la memoria MD (18) y de lectura en la memoria MD (21). Posteriormente se tienen en cuenta los retardos de los elementos FMTL (23) y mES (24).

Después del flanco ascendente de la señal de reloj hay que considerar el retardo de escritura en el banco de registros (26) y en la memoria de datos (22), si es el caso.

Preguntas

En la siguiente pregunta se utiliza el diseño ubicado en el directorio PROC_SERIE. También se utiliza alguno de los programas almacenados en los subdirectorios del directorio “programes”.

8 Obtenga las siguientes métricas cuando el procesador base ejecuta el programa euclides: instrucciones ejecutadas, aritmético-lógicas, . . .

Apéndice 4.1: Formato y Codificación de las instrucciones

Formatos e inmediatos.

En la Figura 4.71 se muestran los campos de una instrucción en los cuatro formatos básicos. El tamaño de una instrucción es fijo e igual a 4 bytes. El código de operación siempre está ubicado en los 7 bits menos significativos. Cuando una instrucción utiliza el contenido de registros o almacena el resultado en un de ellos, la ubicación de los identificadores de registros dentro de los 4 bytes es independiente del formato.

	Campos de la instrucción					
	31 . . . 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0
Formato	7	5	5	3	5	7
R	funct7	rs2	rs1	funct3	rd	CoOp
I	literal12		rs1	funct3	rd	CoOp
S	literal7	rs2	rs1	funct3	literal5	CoOp
U	literal20				rd	CoOp

Figura 4.71 *Formatos básicos de las instrucciones.*

En la Figura 4.72 se muestran los dos formatos adicionales B y J[27]. En la Figura 4.72 se identifica explícitamente cómo se utilizan los bits de los campos literal (imm) para construir el inmediato de la instrucción[28]. Observemos que en los formatos B y J, bits contiguos en un campo literal no lo son al construir el inmediato.

	Formatos de instrucción (FI)																	
	31	. . .	25	24 ...	20	19 ...	15	14 ...	12	11	...	7	6 ...	0				
Formato	7			5		5		3		5			7					
S	imm[11:5]			rs2		rs1		funct3		imm[4:0]			CoOp					
B	imm[12] imm[10:5]			rs2		rs1		funct3		imm[4:1] imm[11]			CoOp					
U	imm[31:12]									rd			CoOp					
J	imm[20]	imm[10:1]			imm[11]	imm[19: 12]				rd			CoOp					
	31	30	. . .	21	20	19 ...	15	14 ...	12	11	...	7	6 ...	0				

Figura 4.72 *Formatos de instrucción.*

27. se diferencian de S y U, respectivamente, en la interpretación de los campos literal
28. El campo literal siempre se interpreta como un entero representado en complemento a dos.

En la Figura 4.73 se muestran los distintos tipos de inmediato (TI) especificando el bit que se utiliza de la instrucción. En todos los casos se replica el bit de signo, el cual siempre es el bit 31 del formato de la instrucción (inst[31]).

	Tipos de inmediato (TI)																			
Formato	31	30	...			...	20	19	...		12	11	10	...	5	4	...	1	0	
I	inst[31]												inst[30:25]			inst[24:21]			inst[20]	
S	inst[31]												inst[30:25]			inst[11:8]			inst[7]	
B	inst[31]											inst[7]	inst[30:25]			inst[11:8]			0	
U	inst[31]	inst[30:20]						inst[19:12]				0								
J	inst[31]							inst[19:12]				inst[20]	inst[30:25]			inst[24:21]			0	

Figura 4.73 *Tipos de inmediato.*

Codificación de las instrucciones.

En las tablas de las siguientes figuras se muestra la codificación del subconjunto de las instrucciones RISC-V utilizadas en la práctica. En la Figura 4.74 se muestra el campo código de operación (CoOp). Cinco valores del campo código de operación se utilizan para codificar subconjuntos de instrucciones mediante otros campos de la instrucción. En todos los casos el siguiente campo que hay que decodificar es “funct3” (Figura 4.75, Figura 4.76, Figura 4.77). Adicionalmente, en el caso de los códigos de operación OP y OP_IMM se utiliza el campo “funct7” o “imm”[29] respectivamente.

En las tablas que se muestran, los bits de un campo se dividen en dos partes para construir la tabla. Los bits más significativos de un campo identifican filas y los bits menos significativos identifican columnas. Cuando se utilizan dos campos distintos, cada campo identifica filas o columnas.

Código de operación (CoOp).

CoOp	bits 4 . . . 2, (los bits 1 y 0 son iguales a 1)							
bits 6 . . . 5	000	001	010	011	100	101	110	111[a]
00	LOAD				OP_IMM[a]	auipc		
01	STORE				OP[a]	lui		
10						reser.		
11	BRANCH	jalr	reser.	jal	SYSTEM	reser.		

Figura 4.74 *Campo de código de operación (CoOp). Los dos bits menos significativos del código de operación (1, 0) tienen el valor 1. El símbolo “reser.” indica un código reservado. Los acrónimos en mayúsculas indican que este código de operación lo utilizan un subconjunto de instrucciones. Para su decodificación hay que analizar otros campos de la instrucción. a: Esta combinación de bits no se utiliza en la codificación de instrucciones de 32 bits.*

29. Solo en algunas instrucciones con código de operación OP_IMM.

Las instrucciones “auipc” y “lui” utilizan el tipo de inmediato U. La instrucción “jalr” utiliza el tipo de inmediato I. La instrucción “jal” utiliza el tipo de inmediato J.

Códigos de operación OP_IMM y OP. Las instrucciones OP utilizan el formato de instrucción denominado R y las instrucciones OP_IMM utilizan el formato de instrucción denominado I. Para identificar las instrucciones que agrupan hay que decodificar los campos “funct3” y “funct7” en las instrucciones OP y el campo “imm(10:4)” en algunas instrucciones OP_IMM (slli, srli, srai). Estos dos últimos campos (funct7 e imm(10:4)) ocupan los mismos bits en la codificación de la instrucción, pero se identifican de forma distinta, debido a que las instrucciones OP_IMM tienen un campo inmediato.

	funct3 (bits 14. . . 12)							
F7IM (bit 5)	000	001	010	011	100	101	110	111
0	addi / add	slli / sll	slti / slt	sltui / sltu	xori / xor	srli / srl	ori / or	andi / and
1	sub					srai / sra		

Figura 4.75 *Códigos de operación OP_IMM u OP. Las instrucciones en la misma casilla se identifican por CoOp distintos. Campos funct3 y funct7 o imm(10:4). En la tabla, los campos “func7” e “imm(10:4)” se identifican como F7IM. El bit de F7IM que identifica instrucciones en una columna es el 5. Los otros bits, correspondientes a este campo, tienen el valor cero.*

Código de operación BRANCH. Las instrucciones BRANCH utilizan el formato de instrucción denominado B. Para identificar las instrucciones que agrupa el código de operación, hay que decodificar el campo “funct3”.

funct3	bits 13 . . . 12			
bit 14	00	01	10	11
0	beq	bne		
1	blt	bge	bltu	bgeu

Figura 4.76 *Código de operación BRANCH. Campo funct3.*

Códigos de operación LOAD y STORE. Las instrucciones LOAD utilizan el formato de instrucción denominado I. Las instrucciones STORE utilizan el formato de instrucción denominado B. Para identificar las instrucciones agrupadas en cada código de operación, hay que decodificar el campo “funct3”.

funct3	bits 13 . . . 12			
bit 14	00	01	10	11
0	lb / sb	lh / sh	lw / sw	
1	lbu	lhu		

Figura 4.77 *Códigos de operación LOAD y STORE. Campo funct3.*

Apéndice 4.2: Clasificación de instrucciones por funcionalidad

En las siguientes figuras se agrupan las instrucciones por funcionalidad o por la forma de obtener los operandos fuentes y se muestra la interpretación de los campos de las instrucciones.

Instrucciones de acceso a memoria. Utilizan un registro para calcular la dirección efectiva y el otro registro contiene el dato que se almacena en memoria o es el registro donde se almacena el dato leído de memoria.

	Formatos de instrucción						
	31 . . . 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0	
Formato	7	5	5	3	5	7	**Campo funct3**
I	imm[11:0]		rs1	funct3	rd	CoOp	lb / lh / lw / lbu / lhu
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	CoOp	sb / sh / sw

Figura 4.78 *Instrucciones de acceso a memoria.*

Instrucciones de cálculo con un operando en la instrucción. Un operando es el contenido de un registro y el otro operando es un literal (offset), especificado en la instrucción, que debe interpretarse en complemento a dos. El resultado se almacena en el banco de registros.

	Formatos de instrucción						
	31 . . . 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0	
Formato	7	5	5	3	5	7	**campo funct3 y I-Imm**
I	imm[11:0]		rs1	funct3	rd	CoOp	addi / slli / slti / sltiu / andi / ori / xori / srli / srai

Figura 4.79 *Instrucciones de cálculo con un operando en la instrucción y el otro operando es un registro.*

Instrucciones de cálculo con operando en registros. Los dos operandos se leen del banco de registros y el resultado se almacena en el banco de registros.

	Formatos de instrucción						
	31 . . . 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0	
Formato	7	5	5	3	5	7	**campo funct3 y func7**
R	funct7	rs2	rs1	funct3	rd	CoOp	add / sub / sll / slt / sltu / and / or / xor / srl / sra

Figura 4.80 *Instrucciones de cálculo con operandos en registros.*

Instrucciones de secuenciamiento con direccionamiento relativo. La dirección de la instrucción destino del salto se calcula de forma relativa a la dirección de la instrucción de secuenciamiento. Además se almacena la dirección de la siguiente instrucción.

	Formatos de instrucción																			Código de operación
	31	. . .	25	24	...	20	19	...	15	14	...	12	11	...	7	6	...	0		
Formato	7			5			5			3			5			7				
	31	30	. . .	21		20	19	...	15	14	...	12	11	...	7	6	...	0		
J	imm[20]	imm[10:1]				imm[11]	imm[19: 12]						rd			CoOp				jal

Figura 4.81 *Instrucciones de secuenciamiento con direccionamiento relativo a la dirección de la instrucción de secuenciamiento.*

Instrucciones de secuenciamiento con direccionamiento indexado. La dirección de la instrucción destino del salto se calcula utilizando el contenido de un registro del banco de registros.

	Formatos de instrucción																			Código de operación
	31	. . .	25	24	...	20	19	...	15	14	...	12	11	...	7	6	...	0		
Formato	7			5			5			3			5			7				
I	imm[11:0]						rs1			funct3			rd			CoOp				jalr

Figura 4.82 *Instrucciones de secuenciamiento con direccionamiento indexado. Para el cálculo de la dirección destino se utiliza el contenido de un registro del banco de registros.*

Instrucciones de secuenciamiento condicional con direccionamiento relativo.

La dirección de la instrucción destino del salto se calcula de forma relativa a la dirección de la instrucción de secuenciamiento. El campo literal (offset) formateado y multiplicado por dos se suma a la dirección de la instrucción de secuenciamiento. Por dirección destino se entiende la dirección de la siguiente instrucción que se interpreta cuando se cumple la condición.

	Formatos de instrucción																		Campo funct3
	31	. . .	25	24	...	20	19	...	15	14	...	12	11	...	7	6	...	0	
Formato	7			5			5			3			5			7			
B	imm[12] imm[10:5]			rs2			rs1			funct3			imm[4:1] imm[11]			CoOp			beq / bne / blt / bge / bltu / bgeu

Figura 4.83 *Instrucciones de secuenciamiento condicional con direccionamiento relativo. El contenido de los registros fuente1 y fuente2 se utiliza para evaluar la condición.*

Apéndice 4.3: Especificación semántica de las instrucciones

Códigos de operación

En las tablas de las siguientes figuras se describen las instrucciones. Se utilizan tres tablas. En la tabla de la Figura 4.84 se muestran las instrucciones que se pueden decodificar directamente a partir del código de operación. Son instrucciones que utilizan los formatos U, J e I. El superíndice v en un identificador de registro indica contenido del registro.

Código de operación	FI	Nemotécnico y especificación	Descripción	Especificación semántica	Campos con valor obligatorio
0010111	U	auipc rd, imm	adder upper immediate to PC	rd^v = CP + (U_Imm << 12)	
0110111	U	lui rd, imm	load upper immediate	rd^v = U_Imm << 12	
1100111	I	jalr rd, rs1, imm	unconditional jump and link register	$CP^v = (rs1^v + (\text{ I-Imm }))\vert_{31:1}$ & '0' y $rd^v = CP^v + 4$	funct3 = 0
1101111	J	jal rd, imm	unconditional indexed jump and link	$CP^v = CP^v$ + (J-Imm << 1) y $rd^v = CP^v + 4$	
1100011	B	BRANCH	conditional branch operation	Figura 4.85	
0000011	I	LOAD	load memory operation	Figura 4.86	
0100011	S	STORE	store memory operation	Figura 4.87	
0010011	I	OP_IMM	integer register-immediate operation	Figura 4.88	
0110011	R	OP	integer register-register operation	Figura 4.89	
		ExtSig: extensión de signo J-Imm = ExtSig (inst(31), inst(19:12), inst(20), inst(30:21)) I-Imm = ExtSig (inst(31:20)		U_Imm = ExtSig(inst(31:12)) << 1: el imm es múltiplo de 2 << 12: desplazar 12 posiciones a la izquierda &: indica concatenación.	

Figura 4.84 *Códigos de operación de las instrucciones y especificación semántica. FI: formato de la instrucción.*

Grupos de instrucciones con el mismo código de operación

Instrucciones de secuenciamiento condicional. En la tabla de la Figura 4.85 se muestran las instrucciones cuyo código de operación es BRANCH y requieren analizar el campo “funct3” para una posterior decodificación. Estas instrucciones utilizan el formato B.

Campo funct3	FI	Nemotécnico y especificación	Descripción	Operación
000	B	beq rs1, rs2, imm	branch equal	if ($rs1^v$ == $rs2^v$) then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
001	B	bne rs1, rs2, imm	branch not equal	if ($rs1^v$!= $rs2^v$) then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
100	B	blt rs1, rs2, imm	branch less than; signed	if ($rs1^v$ < $rs2^v$)$\|_e$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
101	B	bge rs1, rs2, imm	branch greater than or equal; signed	if ($rs1^v$ >= $rs2^v$)$\|_e$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
110	B	bltu rs1, rs2, imm	branch less than; unsigned	if ($rs1^v$ < $rs2^v$)$\|_n$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4
111	B	bgeu rs1, rs2, imm	branch greater than or equal; unsigned	if ($rs1^v$ >= $rs2^v$)$\|_n$ then CP^v = CP^v + (B-Imm <<1); else CP^v = CP^v + 4

ExtSig: extensión de signo
B-Imm = ExtSig (inst(31), inst(7), inst(30:25), inst(11:8))

<< 1: el imm es múltiplo de 2
$\|_e$: comparación de enteros. $\|_n$: comparación de naturales.

Figura 4.85 *Código de operación BRANCH. Códigos del campo funct3. FI: formato de la instrucción.*

Instrucciones de acceso a memoria. En la Figura 4.86 y en la Figura 4.87 se muestran las instrucciones cuyos códigos de operación son LOAD y STORE respectivamente. El formato de instrucción de las instrucciones LOAD es I y el formato de instrucción de las instrucciones STORE es S. Los tipos de inmediato respectivos se identifican con los mismo acrónimos. Estos subconjuntos de instrucciones requieren analizar el campo “funct3” para una posterior decodificación.

Campo funct3	FI	Nemotécnico y especificación	Descripción	Operación
000	I	lb rd, imm(rs1)	load byte	rd^v = ExtSig ((M [$rs1^v$ + I-Imm])$\|_{7:0}$)
001	I	lh rd, imm(rs1)	load halfword	rd^v = ExtSig ((M [$rs1^v$ + I-Imm])$\|_{15:0}$)
010	I	lw rd, imm(rs1)	load word	rd^v = M [$rs1^v$ + I-Imm]
100	I	lbu rd, imm(rs1)	load byte unsigned	rd^v = ExtCero ((M [$rs1^v$ + I-Imm])$\|_{7:0}$)
101	I	lhu rd, imm(rs1)	load halfword unsigned	rd^v = ExtCero ((M [$rs1^v$ + I-Imm])$\|_{15:0}$)

ExtSig: extensión de signo
ExtCero: extensión con ceros

$\|_{b...a}$: secuencia consecutiva de bits
I-Imm = ExtSig (inst(31:20))

Figura 4.86 *Código de operación LOAD. Códigos del campo funct3. FI: formato de la instrucción.*

Notemos que el inmediato en las instrucciones STORE se construye de forma distinta al de las instrucciones LOAD.

Campo funct3	FI	Nemotécnico y especificación	Descripción	Operación
000	S	sb rs2, imm(rs1)	store byte	M [$rs1^v$ + S-Imm] = $rs2^v|_{7:0}$
001	S	sh rs2, imm(rs1)	store halfword	M [$rs1^v$ + S-Imm] = $rs2^v|_{15:0}$
010	S	sw rs2, imm(rs1)	store word	M [$rs1^v$ + S-Imm] = $rs2^v$

ExtSig: extensión de signo
S-Imm = ExtSig (inst(31:25), inst(11:7))

$|_{b..a}$: secuencia consecutiva de bits

Figura 4.87 *Código de operación STORE. Códigos del campo funct3. FI: formato de la instrucción.*

Instrucciones de cálculo. En la Figura 4.88 se muestran las instrucciones cuyo código de operación es OP_IMM. Este subconjunto de instrucciones requiere analizar el campo "funct3" para una posterior decodificación. En tres instrucciones (slli, srli y srai) solo se utilizan los cincos bits menos significativos del campo literal como inmediato (shmat). El resto de bits (I-Imm*) se utiliza como una extensión del campo "funct3". Esto es, hay que decodificarlo.

Campo I-Imm *	Campo funct3	FI	Nemotécnico y especificación	Descripción	Operación
	000	I	addi rd, rs1, imm	add immediate	rd^v = $rs1^v$ add I-Imm
0000000	001	I	slli rd, rs1, shamt	shift left logical inmediate	rd^v = $rs1^v$ << shamt
	010	I	slti rd, rs1, imm	set less than inmediate, signed	if ($rs1^v$ < I-Imm)$|_e$ then rd^v = 1 else rd^v = 0
	011	I	sltiu rd, rs1, imm	set less than inmediate, unsigned	if ($rs1^v$ < I-Imm)$|_n$ then rd^v = 1 else rd^v = 0
	100	I	xori rd, rs1, imm	xor immediate	rd^v = $rs1^v$ xor I-Imm;
0000000	101	I	srli rd, rs1, shamt	shift right logical inmediate	rd^v = $rs1^v$ >>$|_{lo}$ shamt
0100000	101	I	srai rd, rs1, shamt	shift right arithmetic inmediate	rd^v = $rs1^v$ >>$|_{ar}$ shamt
	110	I	ori rd, rs1, imm	or immediate	rd^v = $rs1^v$ or I-Imm
	111	I	andi rd, rs1, imm	and immediate	rd^v = $rs1^v$ and I-Imm

ExtSig: extensión de signo
I-Imm = ExtSig (inst(31:20))

shamt = inst(24:20)
$|_{lo}$: desplazamiento lógico
$|_{ar}$: desplazamiento aritmético

$|_e$: aritmética entera
$|_n$: aritmética con números naturales

Figura 4.88 *Código de operación OP_IMM. Códigos de los campo funct3 e I-Imm* = inst(31:25). FI: formato de la instrucción.*

En la Figura 4.89 se muestra el conjunto de instrucciones con código de operación OP. En estas instrucciones los operando fuentes están almacenado en registros. Este subconjunto de instrucciones requiere analizar el campo" funct3" y el campo "funct7" para una posterior decodificación.

Campo funct7	Campo funct3	FI	Nemotécnico y especificación	Descripción	Operación
0000000	000	R	add rd, rs1, rs2	add registers	$rd^v = rs1^v$ add $rs2^v$
0100000	000	R	sub rd, rs1, rs2	sub registers	$rd^v = rs1^v$ sub $rs2^v$
0000000	001	R	sll rd, rs1, rs2	shift left logical variable	$rd^v = rs1^v << rs2^v\|_{4:0}$
0000000	010	R	slt rd, rs1, rs2	set less than, signed	if $(rs1^v < rs2^v)\|_e$ then $rd^v = 1$ else $rd^v = 0$
0000000	011	R	sltu rd, rs1, rs2	set less than, unsigned	if $(rs1^v < rs2^v)\|_n$ then $rd^v = 1$ else $rd^v = 0$
0000000	100	R	xor rd, rs1, rs2	xor registers	$rd^v = rs1^v$ xor $rs2^v$
0000000	101	R	srl rd, rs1, rs2	shift right logical	$rd^v = rs1^v >>!_{lo} rs2^v\|_{4:0}$
0100000	101	R	sra rd, rs1, rs2	shift right arithmetic	$rd^v = rs1^v >>!_{ar} rs2^v\|_{4:0}$
0000000	110	R	or rd, rs1, rs2	or registers	$rd^v = rs1^v$ or $rs2^v$
0000000	111	R	and rd, rs1, rs2	and registers	$rd^v = rs1^v$ and $rs2^v$

ExtSig: extensión de signo
$|_e$: interpretación como números enteros, codificados en complemento a dos
$|_n$: interpretación como números naturales

$|_{lo}$: desplazamiento lógico
$|_{ar}$: desplazamiento aritmético
$|_{4:0}$: 5 bits menos significativos

Figura 4.89 *Código de operación OP. Códigos de los campo funct3 y funct7. FI: formato de la instrucción.*

Instrucción nop. Una instrucción nop canónica (no operación) se implementa mediante la siguiente codificación.

```
addi x0, x0, #0
```

Instrucción de retorno de procedimiento. Al efectuar la llamada a un procedimiento la dirección de retorno se almacena en un registro. Entonces, el retorno (canónico) se implementa de la siguiente forma, donde el registro rs1 almacena la dirección de retorno.

```
jalr x0, rs1, #0
```

Apéndice 4.4: Control del encaminamiento por instrucción

En la tabla de la Figura 4.90 se describen las señales de encaminamiento para cada instrucción (minúsculas) o subconjunto de instrucciones (mayúsculas). Además, se indican las unidades funcionales utilizadas.

En lugar de una codificación específica se utilizan acrónimos para indicar la función “logica”. Por ejemplo, en la columna PBR se indica que debe actualizarse un registro del banco de registros (si) o no (no). Este mismo acrónimo se utiliza en las columnas opXXX. En este caso, indica que se utiliza la unidad funcional correspondiente. La operación concreta se detalla en el Apéndice 4.6, el Apéndice 4.7 y el Apéndice 4.8. Por ejemplo, en un acceso a memoria se utilizan las unidades funcionales ALU y MD.

CoOp	bits 4 . . . 2						PBR	inmD	mL1	mL2	mES	inmS	mSI	mSIC	opALU	opMD	opSEC
bits 6 . . . 5	000	001	010	011	100	101											
00	LOAD						si	I	reg	inm	MD			secuImpl	si	si	no
					OP_IMM		si	I	reg	inm	ALU			secuImpl	si	no	no
						auipc	si	U	CP	inm	ALU			secuImpl	si	no	no
01	STORE						no	S	reg	inm				secuImpl	si	si	no
					OP		si		reg	reg	ALU			secuImpl	si	no	no
						lui	si	U	cero	inm				secuImpl	si	no	no
10																	
11	BRANCH						no		reg	reg		B	rel	secu	no	no	si
		jalr					si		reg		ret	I	indx	secuMod	no	no	si
				jal			si				ret	J	rel	secuMod	no	no	si
					SYSTEM												

Figura 4.90 *Señales de encaminamiento para las instrucciones e indicación de las unidades funcionales utilizadas.*

Cuando en una casilla no hay ningún acrónimo se está indicando que no es importante el valor.

En las columnas que identifican explícitamente multiplexores se indica la fuente que debe seleccionarse. Por ejemplo para los multiplexores mL1 y mL2: a) banco de registros (reg), valor cero (cero), contador de programa (CP) y campo inmediato formateado (inm). La selección concreta del campo formateado se indica en la columna inmD.

Los elementos FMTD y FTMS se encargan de formatear, de varias formas, algunos bits de la instrucción (campo literal o imm). Para la selección de uno de ellos se utiliza un multiplexor interno. El acrónimo de la tabla indica el formato que debe seleccionarse.

En la Figura 4.91 se muestra la codificación binaria de los acrónimos inmD, mL1, mL2 y mES.

Acrónimo	U	I	S
inmD	00	01	10

Acrónimo	reg	CP	cero
mL1	00	01	10

Acrónimo	reg	inm
mL2	0	1

Acrónimo	ALU	MD	ret
mES	00	01	10

Figura 4.91 *Codificación binaria de los acrónimos inmD, mL1, mL2 y mES.*

En la Figura 4.92 se muestra la codificación de la señal PBR y del bit más significativo de las señales opALU, opMD y opSEC.

Acrónimo	si	no
PBR / opALU / opMD / opSEC	1	0

Figura 4.92 *Codificación binaria de los acrónimos en la señal PBR y bit más significativo de las señales opALU, opMD, opSEC.*

En la Figura 4.93 se muestra la codificación binaria de las señales del camino de datos de secuenciamiento. La codificación del acrónimo “secu” se detalla en el Apéndice 4.8.

Acrónimo	I	B	J
inmS	00	01	10

Acrónimo	rel	indx
mSI	0	1

Acrónimo	seculmpl	secuMod	secu
mSIC	0	1	*

Figura 4.93 *Codificación binaria de los acrónimos en las señales inmS, mSI y mSIC. “*” se detalla en el Apéndice 4.8.*

Apéndice 4.5: Señales de error debido al formato de la instrucción

Cuando el campo “CoOp”, el campo “funct3” o el campo “funct7” (o en el caso de OP_IMM, el campo inst(31:25)) no es válido se activa la señal CoErr. Además, se activa la señal CoErr si alguno de los campos con valor prefijado no es correcto. El primer caso se muestra sombreando en las tablas, la cuadrícula correspondiente (Figura 4.94). En el segundo caso se indican los campos prefijados de forma explícita.

CoOp	bits 4 . . . 2							
bits 6 . . . 5	000	001	010	011	100	101	110	111
00	LOAD				OP_IMM	auipc		
01	STORE				OP	lui		
10								
11	BRANCH			jal	SYSTEM			
		jalr						

Campo prefijado		
2	1	0
funct3 = 0		

Figura 4.94 *Códigos de operación inválidos y campos prefijados para las instrucciones que sólo requieren decodificar el campo código de operación.*

En la Figura 4.95 se muestran de forma sombreada los valores incorrectos del campo “funct3” cuando los códigos de operación son OP_IMM u OP. Cuando el CoOp es OP_IMM el campo F7IM solo se tiene en cuenta en las instrucciones “slli”, “srli” y “srai”.

	funct3 (bits 14. . . 12)							
F7IM	000	001	010	011	100	101	110	111
0000000	addi / add	slli / sll	slti / slt	sltui / sltu	xori / xor	srli / srl	ori / or	andi / and
0100000	sub					srai / sra		

Figura 4.95 *Código de operación OP_IMM u OP. Las instrucciones en la misma casilla se identifican por CoOp distinto. Campos funct3 y funct7 o imm(10:4). En la tabla, los campos “funct7” e “imm(10:4)” se identifican como F7IM. El bit de F7IM que identifica instrucciones en una columna es el 6. Valores inválidos de los campos funct3 y F7IM.*

En la Figura 4.96 se muestran de forma sombreada los valores incorrectos del campo “funct3” cuando el código de operación es BRANCH.

funct3	bits 13 . . . 12			
bit 14	00	01	10	11
0	beq	bne		
1	blt	bge	bltu	bgeu

Figura 4.96 *Código de operación BRANCH. Valores inválidos del campo funct3.*

En la tabla de la Figura 4.97 se muestran de forma sombreada los valores incorrectos del campo “funct3” cuando el código de operación es LOAD o STORE.

load				
funct3	bits 13 . . . 12			
bit 14	00	01	10	11
0	lb	lh	lw	
1	lbu	lhu		

store				
funct3	bits 13 . . . 12			
bit 14	00	01	10	11
0	sb	sh	sw	
1				

Figura 4.97 *Códigos de operación LOAD y STORE. Campo funct3. Valores inválidos del campo funct3.*

Apéndice 4.6: Señales de control de la ALU

En la tabla de la Figura 4.98 se muestran las señales de control de la ALU para cada instrucción (módulo DEC de la Figura 4.49, módulo decoopALU de la Figura 4.50). El bit más significativo es idéntico a la columna especificada en la Figura 4.90. Indica si se utiliza la unidad funcional para ejecutar la instrucción. Los siguientes bits codifican la operación concreta.

En la codificación de los conjuntos de instrucciones OP y OP_IMM se utiliza el mismo valor en el campo “funct3” para codificar la misma operación, sobre los operandos en la entrada de la ALU. Recordemos que el código de operación ha sido utilizado para encaminar y formatear, si es el caso, los operandos hacia las entradas de la ALU.

En el conjunto de instrucciones OP, para codificar la operación en la ALU, se utilizan los bits especificados en los campos “funct7” y “funct3” de la instrucción[30]. El tercer bit en opALU es un bit que se extrae del campo “funct7”: bit 5. En la Figura 4.99 se muestra la codificación.

CoOp	bits 4 . . . 2					
bits 6 . . . 5	000	001	010	011	100	101
00	LOAD					auipc
					OP_IMM	
01	STORE					lui
					OP	
10						
11	BRANCH	jalr		jal		
					SYSTEM	

opALU				
4	3	2	1	0
si	suma			
si	ALU_*			
si	suma			
si	ALU_*			
no				

Figura 4.98 *Señal opALU. Los acrónimos si/no se codifican como 1/0.*

Respecto al subconjunto de instrucciones OP_IMM, en algunas instrucciones hay que utilizar el campo “inst(31:25)”, el cual ocupa los mismos bits que el campo “funct7” en la instrucciones OP. En concreto, como tercer bit de opALU se utiliza el bit inst(30) en las instrucciones “srai”, “slli”, “srli”, el cual tiene el valor 1. En las otras instrucciones se establece el valor cero. De esta forma se corresponde con la codificación de las operaciones indicas en las instrucciones OP.

En varias filas de la Figura 4.98 se utiliza el acrónimo “suma”. Este acrónimo se codificará de la misma forma que una instrucción ADD.

30. En lugar del acrónimo de la instrucción se especifica la codificación de la misma en el formato de la instrucción.

	ALU_*			
CoOp	3	2	1	0
OP_IMM	imm(5) = inst(30)	funct3		
OP	funct7(5)	funct3		

Figura 4.99 *Codificación de los 4 bits menos significativos de opALU para los conjuntos de instrucciones OP y OP_IMM.*

En la Figura 4.100 se muestra como ejemplo la codificación de la señal opALU en algunas instrucciones.

	opALU					
	4	3	2	1	0	
Nem.		funct7(5) / imm(5)	funct3			**acrónimo**
add	1	0	000			suma
sub	1	1	000			resta
sll	1	0	001			despl_izquier
slt	1	0	010			cmp_menor_ent
sltu	1	0	011			cmp_menor_nat
slli	1	1	001			despl_izquier
addi	1	0	000			suma
andi	1	0	111			and_bit_a_bit
srai	1	1	101			despl_derec_aritm
auipc	1	0	000			suma
lui	1	0	000			suma

Figura 4.100 *Ejemplos de codificación de opALU. Los campos funct(7) e imm(5) se indican en la Figura 4.99.*

Apéndice 4.7: Señales de control de la memoria de datos

En la tabla de la Figura 4.101 se muestran las señales de control de MD (memoria de datos) para cada instrucción (módulo DEC de la Figura 4.49, módulo decoopMD de la Figura 4.50). El bit más significativo es idéntico a la columna especificada en la Figura 4.90. Indica si se utiliza la unidad funcional para ejecutar la instrucción. Los siguientes bits codifican la operación concreta.

En los dos conjuntos de instrucciones se utilizan los bits especificados en el campo ““funct3” de la instrucción para codificar la operación. Se añade un bit adicional para distinguir entre el subconjunto LOAD y el subconjunto STORE. Este bit es la señal de permiso de escritura en MD (PMD).

Co	bits 4 . . . 2					
bits 6 . . . 5	000	001	010	011	100	101
00	LOAD					
					OP_IMM	auipc
01	STORE					
					OP	lui
10						
11	BRANCH	jalr		jal		
					SYSTEM	

opMD				
4	3	2	1	0
si	MD_L*			
no				
si	MD_S*			
no				
no				

Figura 4.101 *Señal opMD. Los acrónimos si/no se codifican como 1/0.*

Los dos bits menos significativos del campo “funct3” codifican el tamaño del acceso a memoria y el bit más significativo, para el subconjunto LOAD, indica cuando el valor leído debe interpretarse como entero o natural (Figura 4.102).

	MD_L*, MD_S*			
CoOp	3	2	1	0
LOAD	no	funct3		
STORE	si	funct3		

Figura 4.102 *Codificación de los 4 bits menos significativos de opMD para los conjuntos de instrucciones LOAD y STORE. Los acrónimos si/no se codifican como 1/0.*

Los acrónimos byte, media palabra (media_p) y palabras, granularidad del acceso, se codifican de la forma que se indica en la Figura 4.103.

Acrónimo	byte	media_p	palabra
tam	00	01	10

Figura 4.103 *Codificación de la granularidad del acceso.*

El tipo de acceso (load, store) se codifica de la forma que se indica en la Figura 4.104.

Acrónimo	load	store
PMD	0	1

Figura 4.104 *Codificación del tipo de acceso.*

La interpretación de los bits leídos se codifica de la forma que se indica en la Figura 4.105.

Acrónimo	ent	nat
EnNt	0	1

Figura 4.105 *Codificación de la interpretación de los bits leídos.*

Un acceso a memoria se codifica de la forma que se indica en la Figura 4.106.

Acrónimo	si	no
Ac	1	0

Figura 4.106 *Codificación para indicar acceso a memoria.*

En la Figura 4.107 se muestra la codificación de la señal opMD en las instrucciones de acceso a memoria.

	opMD (4... 0)					
	Ac	PMD	EnNt	tam		
Nem.	4	3	2	1	0	acrónimo
lb	1	0	0	0	0	byte
lh	1	0	0	0	1	media_p
lw	1	0	0	1	0	palabra
lbu	1	0	1	0	0	byte
lhu	1	0	1	0	1	media_p
sb	1	1	0	0	0	byte
sh	1	1	0	0	1	media_p
sw	1	1	0	1	0	palabra

Figura 4.107 *Codificación de opMD para las instrucciones de acceso a memoria.*

Apéndice 4.8: Señales de control del secuenciamiento

En la Figura 4.108 se muestra la codificación del secuenciamiento relativo e indexado.

Acrónimo	rel	indx
mSI	0	1

Figura 4.108 *Codificación del tipo de direccionamiento en instrucciones de secuenciamiento.*

En la Figura 4.109 se muestra la codificación de la señal inmS que controla el formateador FMTS en el camino de secuenciamiento.

Acrónimo	SB	I	UJ
inmS	00	01	10

Figura 4.109 *Codificación del formateo del campo inmediato en instrucciones de secuenciamiento.*

En la tabla de la Figura 4.110 se muestran las señales de control del secuenciamiento para cada instrucción (módulo DEC de la Figura 4.49, módulo decoopSEC de la Figura 4.50). El bit más significativo es idéntico a la columna especificada en la Figura 4.90. Indica si es una instrucción que explicita el secuenciamiento. Los siguientes bits codifican la operación concreta.

Co	bits 4 . . . 2						opSEC				mSIC
bits 6 . . . 5	000	001	010	011	100	101	3	2	1	0	
00	LOAD				OP_IMM	auipc	no	secuImplicto			secuImpl
01	STORE				OP	lui	no	secuImplicto			secuImpl
10											
11	BRANCH						si	secuCond			secu
		jalr		jal			si	secuIncond			secuMod
					SYSTEM						

Figura 4.110 *Señal opSEC y control del multiplexor de secuenciamiento mSIC. Los acrónimos* secuImpl/secuMod *se codifican como 0/1.*

Las instrucciones jalr y jal determinan un secuenciamiento incondicional (secuIncond). Las instrucciones BRANCH determinan un secuenciamiento condicional. En estas instrucciones se utilizan los bits especificados en el campo “funct3” para determinar la condición que debe evaluarse (secuCond). El acrónimo “secuInCond”, que indica modificar el secuenciamiento de forma incondicional, se codifica utilizando una combinación libre del campo “funct3” (011).

En la tabla de la Figura 4.111 se muestran las condiciones que se evalúan en las instrucciones BRANCH y el control del multiplexor mSIC (acrónimo secu).

funct3	bits 13 . . . 12			
bit 14	00	01	10	11
0	beq			
		bne		
1	blt			
		bge		
			bltu	
				bgeu

opSEC			
3	2	1	0
si	funct3		

Salidas de EVAL
ig, me, meu
ig
not ig
me
not me or ig
meu
not meu or ig

mSIC (secu)	
cumple	no cumple
secuMod	secuImpl
secuMod	secuImpl
secuMod	secuImpl
secuMod	secuImpl
secuMod	secuImpl
secuMod	secuImpl

Figura 4.111 *Instrucciones BRANCH. Condición que se evalúa y control del multiplexor mSIC. Igualdad: ig, menor utilizando aritmética entera (signed): me, menor utilizando aritmética con naturales (unsigned): meu. Los acrónimos* secuImpl/secuMod *se codifican como 0/1.*

Apéndice 4.9: Organización de los ficheros: árbol de directorios

El árbol de directorios, que contiene los ficheros VHDL, que describen el procesador se muestra desde la Figura 4.112 hasta la Figura 4.116.

En la Figura 4.112 se muestra el árbol de directorios correspondientes a los elementos MI y MD. Los directorios ENSAMBLADO_*/CODIGO contienen los ficheros con la descripción estructural de un elemento. Los directorios *_pkg contienen “packages” que incluyen declaraciones de constantes, tipos y componentes.

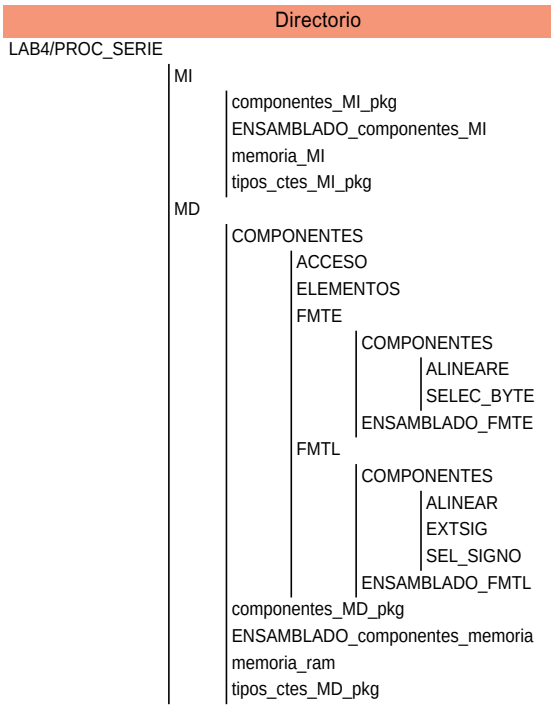


Figura 4.112 *Arbol de directorios de los elementos MI y MD.*

En la Figura 4.113 se muestra el árbol de directorios correspondiente al procesador, el cual incluye el autómata de control, el camino de datos y el registro CP. En el árbol de camino de datos se distingue la unidad de cálculo (UC) y la unidad de secuenciamiento (USE). Los directorios ENSAMBLADO_*/CODIGO contienen la descripción estructural de un elemento.

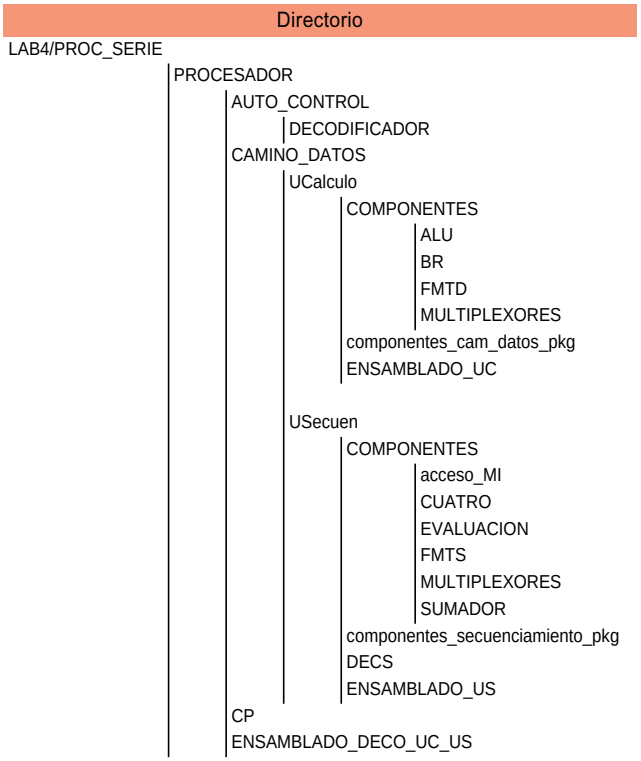


Figura 4.113 *Arbol de directorios del procesador.*

En el directorio ENSAMBLADO/CODIGO está ubicado el fichero VHDL que contiene una descripción estructural del procesador con las memorias (Figura 4.114). En el directorio ENSAMBLADO/PRUEBAS está ubicado el fichero de prueba para la simulación en ModelSIm. En el directorio ENSAMBLADO/QUARTUS están ubicados los fichero *.qpf y *.qsf que utiliza Quartus para efectuar una compilación RTL del diseño. En el fichero *.qsf se describe la ubicación de los ficheros utilizados en el diseño y los ficheros que se utilizan cuando se activa una simulación con ModelSim desde Quartus.

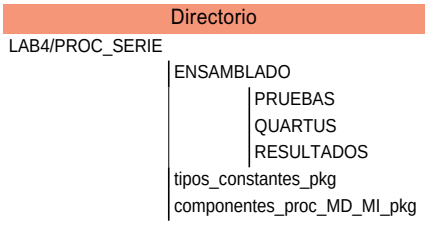


Figura 4.114 *Directorio que contiene los directorios y ficheros describen el ensamblado del procesador con las memorias.*

En la Figura 4.115 se muestra el contenido del directorio “tipos_constantes_pkg”, el cual contiene ficheros con declaraciones de tipos y constantes utilizadas en el diseño y en la simulación del mismo.

Directorio	Fichero	descripción
tipos_constantes_pkg		
	param_disenyo_pkg,vhd	Parámetros de tamaño del banco de registros, memorias y camino de datos
	cte_tipos_deco_camino_pkg.vhd	Subtipos y constantes utilizadas en el control del camino de datos y de las unidades funcionales (U.F.)
	cte_tipos_UF_pkg.vhd	Constantes que codifican la operación en las U.F.
	riscv32_coop_funct_pkg.vhd	Constantes para identificar valores en los campos de una instrucción
	retardos_*.vhd	Especificación de los retardos de los componentes del camino de datos

Figura 4.115 *Ficheros ubicados en el directorio tipos_constantes_pkg.*

En la Figura 4.116 se muestra la ubicación de “packages” que se utilizan en el programa de prueba para mostrar la evolución de la simulación.

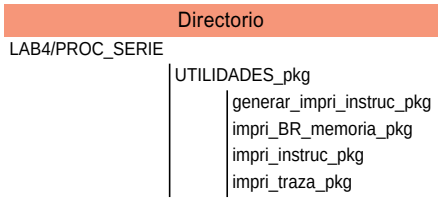


Figura 4.116 *Directorio que contiene “packages” utilizados en el programa de prueba.*

Apéndice 4.10: Esquemas RTL de módulos del procesador

El esquema RTL del procesador se muestra en la Figura 4.117.

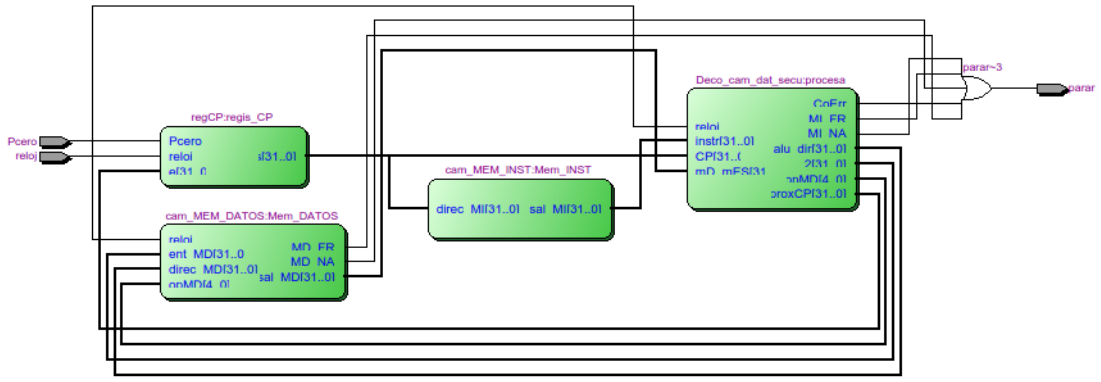


Figura 4.117 *Esquema RTL del procesador.*

En la Figura 4.118 se muestra el esquema RTL del módulo “Deco_cam_dat_secu”, el cual incluye la UC y la USE.

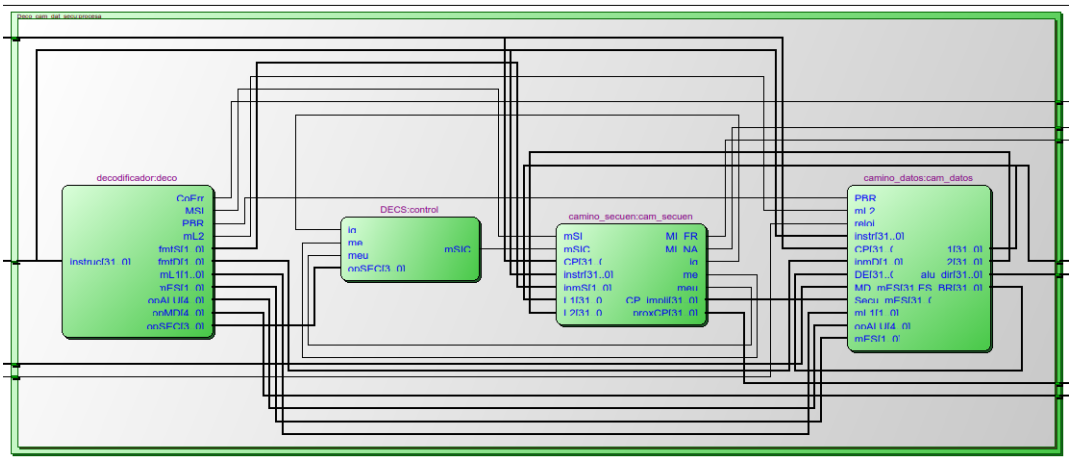


Figura 4.118 *Esquema RTL del conexionado de la UC y USE.*

En la Figura 4.119 se muestra el esquema RTL del módulo cam_MEM_DATOS, el cual contiene la MD y los módulos FMTE, FMTL y acceso_MD.

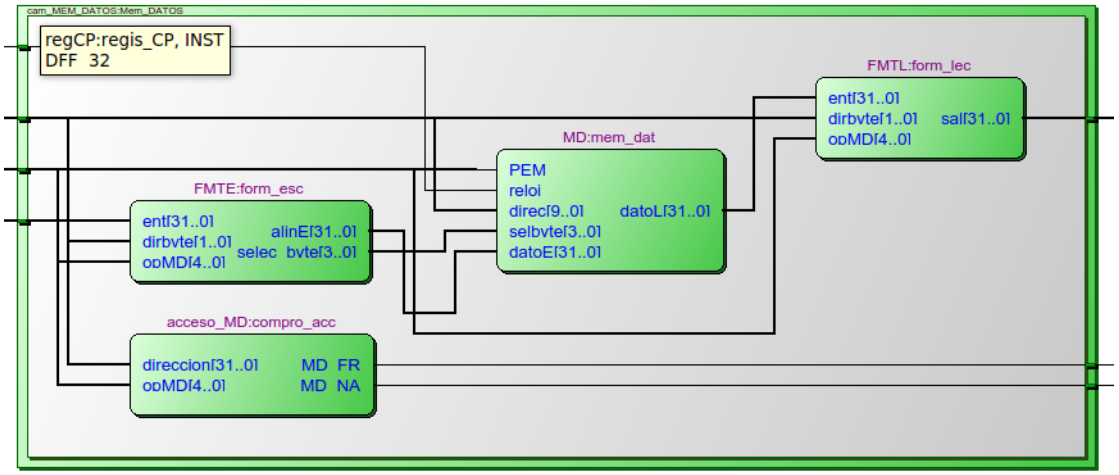


Figura 4.119 *Esquema RTL de la MD y lógica asociada.*

En la Figura 4.120 se muestra el esquema RTL del módulo camino_secuen, el cual incluye la USE.

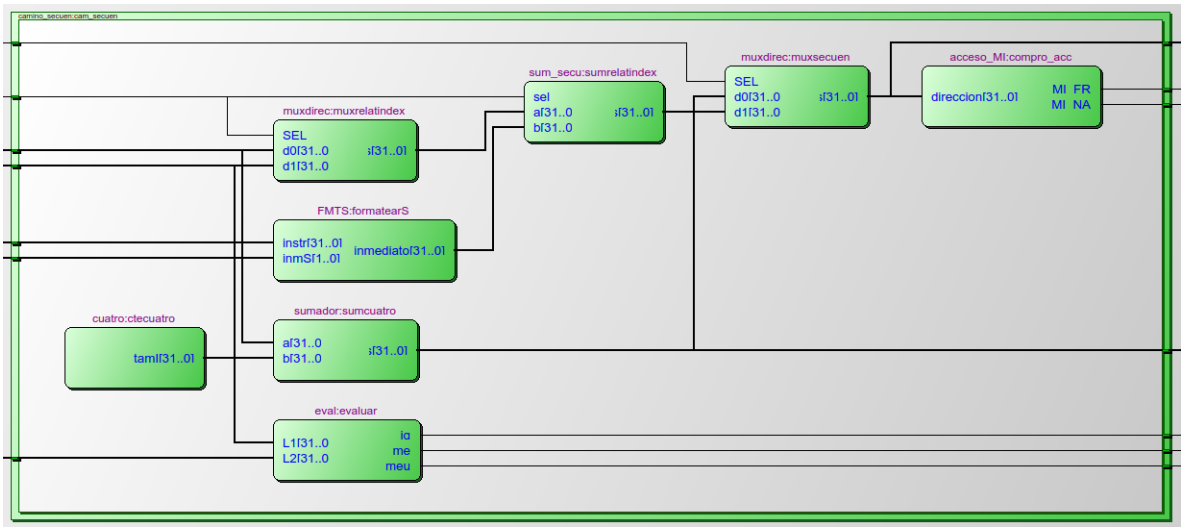


Figura 4.120 *Esquema RTL de la USE.*

En la Figura 4.121 se muestra el esquema RTL del módulo camino_datos (UC), el cual incluye el BR y la ALU junto con multiplexores para el encaminamiento.

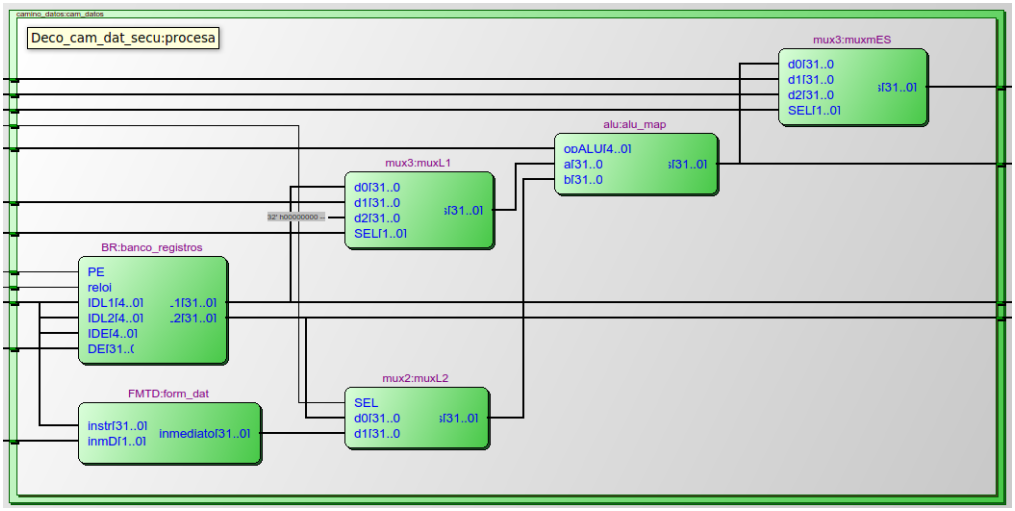


Figura 4.121 *Esquema RTL de la UC.*

Apéndice 4.11: Retardos

En el “package retardos” se declaran los retardos de los elementos del procesador segmentado (Figura 4.122) .

BR	constant retdecoBR: time := 2 ns;
	constant retescBR: time := 5 ns;
	constant retlecBR: time := 5 ns;	

DECS, secu	constant retDECS: time := 2 ns;	
FMTD	constant retfmtD: time := 1 ns;	
FMTE	constant retFMTE: time := 2 ns;	
FMTL	constant retFMTL: time := 2 ns;	
FMTS	constant retfmtS: time := 1 ns;	

MD	constant retdecoMD: time := 2 ns;
	constant retescMD: time := 8 ns;
	constant retlecMD: time := 8 ns;	

MI	constant retdecoMI: time := 2 ns; constant retlecMI: time := 8 ns;	
acceso_MD	constant retexcepMD: time := 1 ns;	se especifican en retardos_even_pkg
acceso_MI	constant retexcepMI: time := 1 ns;	
decp_excep	constant retlogicaexcep: time := 2 ns	
ALU	constant retALU: time := 4 ns;	
camino_datos	constant retMUX3: time := 1 ns; constant retMUX2: time := 1 ns;	
camino_secuen	constant retMUX2: time := 1 ns;	
sumador, sum_secu	constant retSUMADOR: time := 2 ns;	
decodificador	constant retDECO: time := 4 ns;	
eval	constant reteval: time := 1 ns;	
regCP	constant retREGCP: time := 1 ns;	

Figura 4.122 *Retardos de los componentes.*

Apéndice 4.12: Documentación

La documentación ha sido generada utilizando la herramienta Doxygen. El fichero que hay que abrir con un navegador es “index.html” ubicado en el directorio LAB4/PROC_SERIE/documentacio/hmtl.

Las pestañas que se muestran son autoexplicativas.

Dado un módulo se muestra un grafo de dependencias jerárquicas con otros módulos. Pulsando en un nodo del grafo se observan el módulo o módulos que agrupa.

También se puede acceder al código VHDL que describe a un módulo.